

# Multiplatformní webové rozhraní pro zobrazování výrobních dat v reálném čase s možností uživatelských vstupů

## Multi-Platform Web Interface for Visualizing Production Data in Real Time with the Possibility of User Inputs

## Zadání diplomové práce

Student: **Bc. Tomáš Bureš**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Multiplatformní webové rozhraní pro zobrazování výrobních dat v reálném čase s možností uživatelských vstupů  
Multi-Platform Web Interface for Visualizing Production Data in Real Time with the Possibility of User Inputs

Jazyk vypracování: čeština

### Zásady pro vypracování:

Cílem práce je vytvořit webový systém, který poběží jak na velkoplošných obrazovkách, tak na průmyslových tabletech. Systém bude umět komunikovat s připojeným hardware (čtečky, RFID, digitální vstupy.) Systém bude sloužit pro čtení a odvádění dat z výroby a monitorovacích systémů v reálném čase.

1. Popsání vzájemného využití moderních technologií: HTML5, CSS3, AngularJS, TypeScript, NodeJS, SignalR 2.0, ASP.NET MVC.
2. Laboratorní testy limitů technologie SignalR 2.0.
3. Vývoj HTML editoru pro tvorbu obrazovek skládajících se ze základních stavebních jednotek - widgetů.
4. Vytvoření a nakonfigurování ovládacích prvků a obrazovek.
5. Testování v reálném provozu.

### Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.


Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Petr Zima**

Konzultant diplomové práce: Ing. Michal Radecký, Ph.D.

Datum zadání: 01.09.2015

Datum odevzdání: 28.04.2017

  
doc. Dr. Ing. Eduard Sojka  
vedoucí katedry



  
prof. RNDr. Václav Snášel, CSc.  
děkan fakulty

## **Prohlášení studenta**

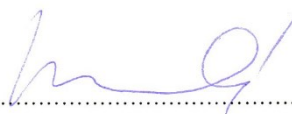
Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 25.4.2017

.....  
Beres

## Prohlášení zástupce spolupracující právnické nebo fyzické osoby

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských/magisterských programech VŠB-TU Ostrava.



.....

V Ostravě 25.4.2017

**SCADA Servis s.r.o.**  
Čedičová 1378/6  
710 00 Ostrava-Slezská Ostrava  
IČ: 26791285. DIČ: CZ26791285

## **Poděkování**

Touto formou bych chtěl poděkovat vedoucímu práce Ing. Petru Zimovi za vedení při psaní diplomové práce. Také bych chtěl poděkovat mému zaměstnavateli Ing. Romanu Krusberskému za celkové umožnění práce na projektu. V neposlední řadě bych chtěl poděkovat panu Ing. Michalovi Radeckému Ph.D., který byl mým konzultantem a pomohl mi nasměrovat strukturu práce.

## **Abstrakt**

V této práci se zabývám kompletním návrhem a popisem implementace webového informačního systému. V začátku práce popisuji samotné zadání práce formou specifikace požadavků. V další části se zaměřuji na analýzu zadání a popisuji návrh celé aplikace. Po dokončení návrhu systému popisuji jeho samotnou implementaci. V závěru práce potom hodnotím vytvořený projekt a zobrazuji snímky obrazovky z již hotové aplikace nasazené u několika zákazníků. Neopomněl jsem také nedostatky aplikace a směr, kam se posunout dále.

**Klíčová slova:** Web, Webová aplikace, Angular JS, SignalR, WebSocket, ASP.NET

## **Abstract**

In this thesis I deal with the complete design and description of the implementation of web information system. At the beginning of the thesis I describe the assignment itself in a form of requirements specification. In the next part, I focus on analyzing the assignment and describing the design of the whole application. After completing the design of the system I describe the implementation itself. At the end of my thesis I evaluate the created project and display screenshots from the already finished application deployed to several customers. I also did not forget the shortcomings of the application and direction in which to move further.

**Keywords:** Web, Web application, Angular JS, SignalR, WebSocket, ASP.NET

## Seznam použitých zkratk a pojmů

API	Rozhraní pro programování aplikací
ASP.NET	Framework pro tvorbu webových aplikací [1]
CRUD	Vytvoření, čtení, aktualizace a smazání (create, read, update, delete)
CSS	Kaskádové styly (cascade style sheets)
HTML	Značkovací jazyk s podporou odkazů (hypertext markup language)
HW	Hardware
ID	Identifikace
IE	Internet explorer
IIS	Internetová informační služba
JSON	Datový formát (JavaScript Object Notation)
LCD	Displej z tekutých krystalů (liquid crystal display)
MVC	Softwarová architektura (model view controller)
PC	Počítač
RFID	Identifikace na rádiové frekvenci (Radio Frequency Identification)
SVG	Škálovatelná vektorová grafika (Scalable vector graphics)
URL	Jednotná adresa zdroje (Uniform Resource Locator)
XAF	Express application framework
XML	Rozšířitelný značkovací jazyk (eXtensible Markup Language)
XPO	Express persistent object
ZIP	Souborový formát
ms	milisekundy
px	pixely

## Obsah

1. Úvod.....	11
2. Specifikace požadavků.....	12
2.1. Úvod.....	12
2.1.1. Účel .....	12
2.1.2. Rozsah systému .....	12
2.1.3. Zkratky .....	12
2.2. Všeobecný popis .....	12
2.2.1. Kontext produktu.....	12
2.2.2. Přehled funkcí .....	12
2.2.3. Profil uživatele .....	12
2.2.4. Přehled omezujících podmínek .....	13
2.2.5. Předpoklady a závislosti.....	13
2.3. Specifikace požadavků.....	13
3. Návrh řešení .....	14
3.1. Webový server.....	14
3.2. Základní komponenty.....	14
3.3. Zobrazení dat ze serveru .....	15
3.4. Skripty a styly .....	16
3.5. Javascript API .....	17
3.6. Práce se stavy .....	17
3.6.1. Multi state.....	17
3.6.2. Array state .....	18
3.6.3. Treshold state .....	18
3.7. Události widgetu .....	18
3.8. Souborový systém .....	18
3.9. Vlastnosti Editoru.....	18
3.10. Design Editoru.....	19
3.11. Ukládání dat .....	20
3.12. Základní widgety.....	21
3.13. Komunikace s periferiemi .....	22
3.14. Security .....	23
3.15. Inicializace komunikace.....	24



4.	Testování SignalR .....	26
4.1.	Počet připojení .....	26
4.2.	Množství dat.....	26
5.	Implementace .....	28
5.1.	Služby.....	28
5.1.1.	ApplicationStatus .....	28
5.1.2.	Authentication .....	29
5.1.3.	ContextMenu.....	29
5.1.4.	CustomContent.....	30
5.1.5.	DataLayer.....	31
5.1.6.	Dialog.....	31
5.1.7.	Editor.....	33
5.1.8.	Evaluator .....	34
5.1.9.	Hotkeys .....	34
5.1.10.	Language .....	34
5.1.11.	Logger .....	35
5.1.12.	ScreenScale .....	35
5.1.13.	SignalR.....	35
5.1.14.	SocketIO.....	37
5.1.15.	Theme.....	37
5.1.16.	Toolbox .....	37
5.1.17.	Transport .....	38
5.1.18.	VariableLayer.....	38
5.1.19.	WebClient.....	39
5.1.20.	Widgets .....	40
5.2.	Model Aplikace.....	41
5.3.	Property .....	42
5.4.	Vlastnosti.....	43
5.5.	Editace.....	44
5.6.	Jádro aplikace.....	46
5.6.1.	Lokální proměnné .....	46
5.6.2.	Globální a serverové proměnné.....	47
5.6.3.	Aktualizace.....	47

6.	Nasazení aplikace .....	49
6.1.	Přehledové obrazovky .....	49
6.2.	Odvádění výroby .....	49
6.3.	Sledování technologie .....	51
6.4.	Mobilní terminály.....	51
7.	Nedostatky, kam to posunout dál .....	53
7.1.	Problémy s výkonem.....	53
7.2.	Nápady na vylepšení .....	54
8.	Závěr .....	55
9.	Reference.....	56
Příloha A.	Obsah přiloženého DVD .....	58

# 1. Úvod

Na úvod bych chtěl napsat, že pracuji ve firmě, která se zabývá sledováním a optimalizací výroby. Zadání této práce bylo podmíněno potřebou vytvořit univerzální webové rozhraní pro moderní dotykové zařízení i pro velkoplošné informační zobrazení. Tyto terminály budou umístěny ve výrobních halách a budou sbírat informace ze strojů a od operátorů.

Prvním krokem ve vývoji aplikace je získat veškeré požadavky na aplikaci. Jelikož bude aplikace sloužit jak odbornému personálu a programátorům, ale i operátorům ve výrobě, bylo třeba tyto informace získat z více míst. Některé požadavky byly od nás, jakožto programátorů, další od manažerů výroby, lokálních administrátorů a operátorů výroby.

Na základě těchto požadavků jsem vytvořil návrh aplikace. Před začátkem vývoje jsem musel zvolit technologie, které se budou používat. Jelikož byl projekt založen již před několika lety, neměl jsem dostatek zkušeností, a proto některé technologie nebyly zvoleny zcela dobře. V závěru práce potom tyto chyby rozeberu. Bylo třeba se zaměřit i na serverovou část a definovat nějaký standard komunikace. Dále jsem musel vyřešit problém komunikace webové stránky s hardwarem terminálu, konkrétně RFID čtečky, nebo čtečky čárových kódů. Systém v průběhu let prodělal několik změn, ať v systému jádra funkcionality, designu, nebo formátu ukládání dat.

V závěru práce se poté zaměřím na ukázky již hotové aplikace. V ukázce uvidíme nasazení aplikace v různých scénářích. Například přehledové obrazovky, odvádění ve výrobě, potrubní rozvody a ukázka mobilního terminálu.

## 2. Specifikace požadavků

### 2.1. Úvod

V současné době nemá mnoho výrobních podniků žádný způsob online sledování výroby. Co se týká odvádění, tak ve většině případech musí operátoři vyplňovat různé tištěné dokumenty a ty se následně musí přepisovat do digitální podoby. Tímto přepisem mohou vznikat chyby a my se budeme snažit tyto přepisy odstranit – tzv. bezpapírová výroba.

#### 2.1.1. Účel

Aplikace bude vytvořena za účelem zrychlení a ulehčení tvorby nových webových projektů. Bude umožňovat provádět úpravy na systému i neodbornému personálu. Aplikace bude sloužit k univerzální tvorbě webových náhledů na aktuální nebo historická data. Data budou zobrazována staticky, nebo dynamicky na základě uživatelských akcí nebo změn v systému. Editace bude probíhat v editačním režimu, který bude přístupný na základě oprávnění. V editačním režimu bude možné tvořit stránky z již existujících prvků, dále nazývány jako "widgety". Také bude možné vlastní widgety vytvářet.

#### 2.1.2. Rozsah systému

Aplikace bude obsahovat

- editor
- sadu základních widgetů
- jednoznačně definované komunikační rozhraní se serverem
- integrovanou uživatelskou dokumentaci

#### 2.1.3. Zkratky

- Widget – jedná se o základní stavební prvek aplikace. Každá stránka je sestavena z několika takových widgetů. Widget se může skládat z dalších widgetů.
- HW – hardware
- HTML – hypertext markup language [2]
- CSS – cascade style sheets [3]

## 2.2. Všeobecný popis

### 2.2.1. Kontext produktu

Celá aplikace bude moct fungovat samostatně, ale primárně se bude využívat jako modul firemního systému ImproveIT. V případě využití samostatného nasazení, nemusí být dostupné všechny funkce aplikace. Pro zobrazení dat bude aplikace potřebovat aplikační server. S tím to serverem bude komunikovat například za pomoci websocket protokolu.

### 2.2.2. Přehled funkcí

- Tvorba stránek a widgetů
- Zobrazování aktuálních a historických dat
- Komunikace s HW (čtečky čárových kódů, čtečky RFID karet, váhy...)

### 2.2.3. Profil uživatele

Aplikaci budou využívat tři skupiny uživatelů.

- Vývojáři – Budou využívat veškeré možnosti aplikace v editačním režimu, editace skriptů, tvorba stránek a widgetů.
- Administrátoři – Budou mít přístup do editačního režimu s možností editace vlastností widgetů. Například zobrazovaný text, barvu pozadí a podobně.
- Běžný uživatel – Nebude mít přístup do editačního režimu. Bude si prohlížet zobrazovaná data, případně jinak pracovat s naskriptovaným systémem

#### 2.2.4. Přehled omezujících podmínek

Při tvorbě aplikace je nutné myslet na to, že editaci bude provádět i uživatel s pouze základní znalostí znalosti javascriptu, HTML a CSS. Uživatelské rozhraní tomu musí být přizpůsobeno. Systém je třeba testovat na všech moderních prohlížečích Chrome, Firefox, IE10, Edge. Také je třeba zohlednit užívání systému jak za použití klasické myši, tak dotykových obrazovek.

#### 2.2.5. Předpoklady a závislosti

Pro správnou funkčnost HW periférií, jako jsou například čtečky (RFID nebo čárových kódů), je třeba zajistit komunikaci s HW počítače, na kterém aplikace poběží. Pro takovou komunikaci lze využít například Node.js [4] s modulem socket.io [5].

### 2.3. Specifikace požadavků

- Aplikace bude pracovat online ve webovém prostředí
- Aplikace bude podporovat moderní webové prohlížeče. Chrome, Firefox, IE10, Edge
- Aplikace se bude používat ve dvou režimech, editační a prezenční. Editací režim bude sloužit pro tvorbu a úpravu stránek a widgetů. Prezenční režim bude sloužit běžným uživatelům k užívání aplikace.
- Aplikace bude podporovat anonymní a zabezpečený přístup

## 3. Návrh řešení

### 3.1. Webový server

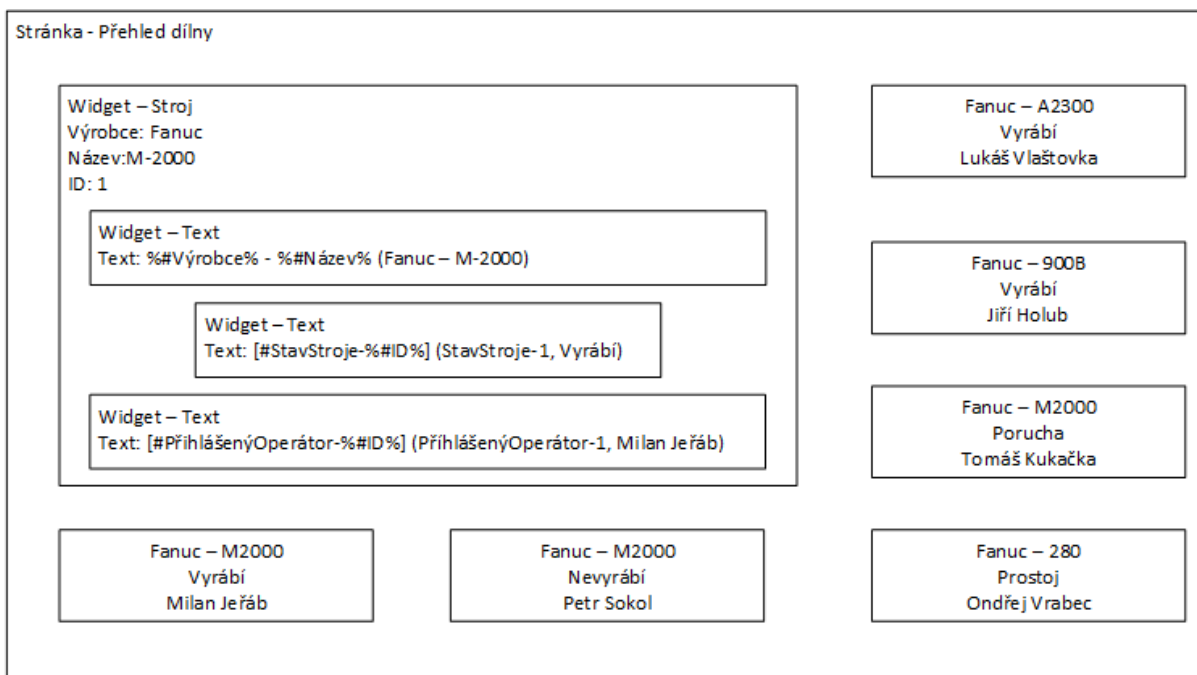
Základem bude webový server, který bude celou aplikaci poskytovat. Pro dobrou podporu nových webových technologií, jako je například komunikace přes websockets [6], je třeba zvolit adekvátní server, který tuto komunikaci podporuje. Pro naši aplikaci bude ideální Windows server 2012 a novější. Jako webový server potom využijeme IIS. Webová aplikace poté poběží na technologii ASP.NET. Jestli zvolíme MVC, nebo starší WebForms, je v podstatě jedno, jelikož aplikace bude využívat pouze několik málo webových metod. Nicméně vzhledem k modernější architektuře zvolíme technologii MVC.

### 3.2. Základní komponenty

Systém bude obsahovat dva základní komponenty. Budou jimi stránka, která se dá chápat jako nějaký kontejner, do které se budou umísťovat další vnořené komponenty, kterým budeme říkat widget. Tyto widgety budou reprezentovat základní zobrazovací prvky, jako například widget typu text, který nám umožní uživateli zobrazit textovou informaci, nebo widget typu rectangle, který bude vykreslovat předdefinovaný obdélník, nebo typ HTML, do kterého se bude dát zapsat jakýkoliv validní HTML. Každý z widgetů bude mít své vlastnosti. Některé z těchto vlastností budou obecná pro všechny widgety, například pozice, šířka, výška, jestli je viditelný a podobně. Další vlastnosti potom budou jedinečné pro každý z widgetů. Widget typu text bude mít vlastnost text, která bude reprezentovat zobrazovaný text. Widget typu rectangle bude mít potom například vlastnosti šířku rámečku, barvu pozadí a podobně.

Systém poté umožní tvořit stránky, do kterých se budou umísťovat již vytvořené widgety. Také nabídne možnost tvorby nových widgetů. Nový widget se poté bude chovat stejně jako stránka, tedy jako kontejner pro další widgety. Nově vytvořený widget může mít statické hodnoty všech svých widgetů, nebo bude umožňovat použít zástupný znak, který umožní vytvořit novému widgetu vlastnost, která se při jeho použití bude dát konfigurovat. Zástupné znaky budou třech typů. První bude reprezentovat lokální proměnné "%#Lokální Proměnná%", globální proměnné "{#Globální Proměnná}" a serverové proměnné "[#Serverová Proměnná]".

Příklad: Chceme vytvořit stránku s názvem přehled dílny. Pro takovou stránku budeme potřebovat widget typu stroj, který bude reprezentovat jednotlivé stroje v dílně. Pro tvorbu tohoto widgetu použijeme tři widgety typu text. První text bude zobrazovat hlavičku stroje. Zápis pro tento text bude vypadat například takto. %#Výrobce% - %#Název%.



Obrázek 1 - Návrh přehledu dílny

Pro zobrazení aktuálního stavu stroje se budeme muset zeptat serveru. Aby server věděl, pro jaký stroj nám má poslat stav, musíme tento stav přidat do dotazu, výsledný dotaz by měl vypadat takto. [#StavStroje-1]. V takovém případě by nám ale server vrátil pro všechny instance widgetů stejný stav. Proto musíme přidat každému widgetu ještě jeho identifikaci. [#StavStroje-%#ID%]. Podobně zajistíme zobrazení jména aktuálně přihlášeného operátora na každém stroji. [#PřihlášenýOperátor-%#ID%]. Pro nově přidávané instance widgetů bude stačit vyplnit pouze výrobce, název a id stroje.

### 3.3. Zobrazení dat ze serveru

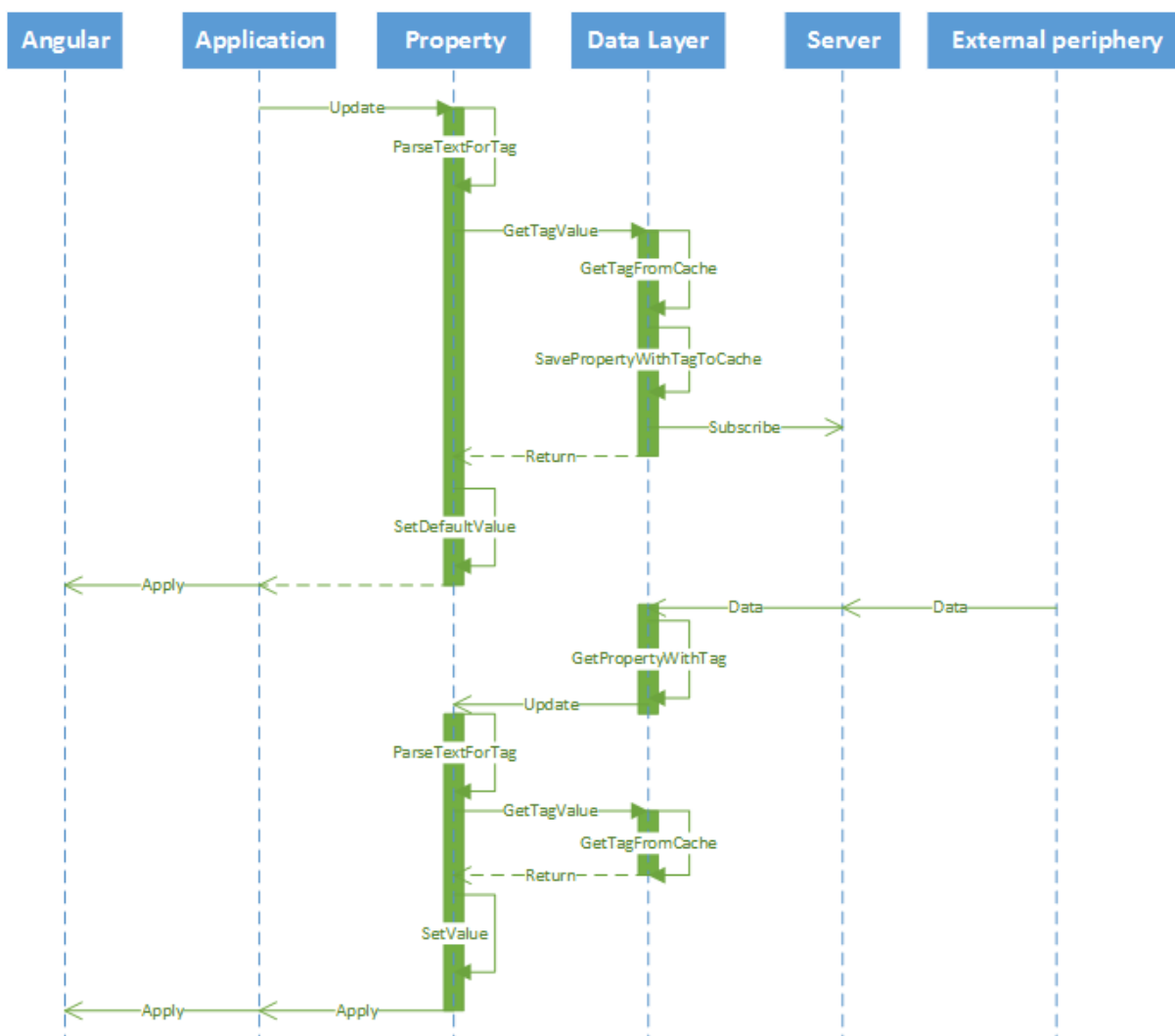
Každý z widgetů bude umožňovat zobrazovat data ze serveru. Pro zobrazení těchto dat je třeba v odpovídající vlastnosti použít syntaxi v tomto tvaru.

[#IdentifikaceDat]

Tato syntaxe zajistí, že si aplikace zaregistruje všechny změny dat pod zadaným názvem. Pokaždé, když server usoudí, že došlo ke změně dat, pošle je na klienta. Data mohou obsahovat jednoduché typy, nebo komplexní objekty. Pokud obdržený objekt bude obsahovat vlastnost Value, zobrazí se hodnota z této vlastnosti, jinak se celý objekt serializuje do stringu. Pro přístup k jednotlivým vlastnostem bude systém podporovat následující syntaxi.

[#IdentifikaceDat->LokálníProměnná.VnořenáProměnná]

V některých případech se může stát, že data ze serveru nebudou hned dostupná, proto bude možnost zadat výchozí hodnotu, která se v případě poruchy komunikace zobrazí.



Obrázek 2 - Zobrazení dat ze serveru

### 3.4. Skripty a styly

Pro složitější systémy bude třeba, aby systém podporoval tvorbu scriptů a stylů. Tyto soubory budou editovatelné z klienta, za pomoci editoru s podporou syntaxe příslušného jazyka. Scripty a styly se při inicializaci aplikace, případně při jejich změně, přidají do kontextu webového prohlížeče, takže je možnost se na ně odkudkoliv odkazovat.

Každá z vlastností jakéhokoliv widgetu, která bude obsahovat zástupný znak, požadavek na data ze serveru, nebo nově odkaz na globální proměnnou, bude mít možnost po změně hodnoty spustit vykonání skriptu. V takovém případě se celý výraz před vyhodnocením nechá zpracovat Javascript funkcí eval.

Zápis do globální proměnné pomocí API: `WebClientAPI.variable.setValue("VariableName", "value");`

Syntaxe globální proměnné: `{#Název Proměnné}`

Příklad. Ze serveru mi v odpovědi na data přijde list přihlášených uživatelů, ale já chci zobrazit jenom jméno prvního z nich a v závorce informaci o celkovém počtu uživatelů. V takovém případě si ve



skriptech vytvořím metodu `getLoggedUsers(users)`, která vrátí požadovaný textový výstup. Syntaxe potom bude vypadat následovně.

```
getLoggedUsers([#LoggedUsers]);
```

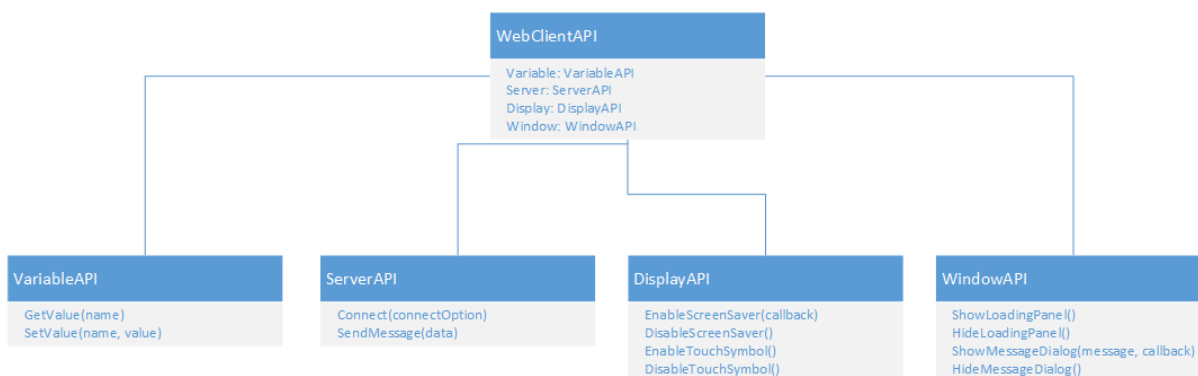
Nebo budu chtít zobrazit objem nádrže, ale ze serveru dostávám informaci pouze o výšce hladiny. Obsah podstavy ale znám, proto můžu bez problémů objem zobrazit.

```
[#TankLevel] * 1000 + ' M3'
```

### 3.5. Javascript API

Pro komunikaci s javascriptem bude systém implementovat jednoduché API. API bude rozděleno do několika skupin podle funkcionality.

- WebClientAPI.Variable – práce s globálními proměnnými
- WebClientAPI.Server – komunikace se serverem, zápis a získávání dat
- WebClientAPI.Display – nastavení spořiče, zobrazení dotykového symbolu
- WebClientAPI.Window – zobrazení pomocných oken a dialogů



Obrázek 3 - WebClientAPI

### 3.6. Práce se stavy

Pro běžné uživatele, kteří hlouběji neovládají javascript, bude systém podporovat definici stavů objektu na základě dat ze serveru. Tyto stavy mohou být trojího typu.

#### 3.6.1. Multi state

V případě tohoto typu musí uživatel definovat, kolik bude mít objekt vstupních proměnných. Každá z těchto proměnných musí nabývat hodnot 0,1 nebo true, false. Po definici počtu se uživateli vygenerují všechny kombinace, které mohou nastat. Podporovaný počet vstupních proměnných je 0 až 5. Kde 0 znamená vypnuto. Počet kombinací je potom  $2^i$  kde  $i$  je počet vstupních proměnných.

Příklad. Vstupy: `[#Data1]`, `[#Data2]`

```

0 0 : color = "red"
0 1 : color = "blue"
1 0 : color = "green"
1 1 : color = "black"

```

Po obdržení Data1 a Data2 se zpracují jejich hodnoty a vyhodnotí se příslušný stav. Pokud Data1 = 1 a Data2 = 0, změní se barva widgetu, v kterém je stav definován na zelenou.

V případě, že by počet možných kombinací (až 32) nestačil, nebo by se barvy v kombinacích hodně opakovaly, lze využít další ze stavů (Array state).

### 3.6.2. Array state

Tento typ má pouze jednu proměnnou. Uživatel si potom může přidávat stavy, které může proměnná nabývat. Stavy začínají na hodnotě 0. Vstupní proměnná se stejně, jako všechny ostatní vlastnosti v aplikaci může nechat zpracovat javascriptem. V takovém případě může syntaxe vstupní proměnné vypadat následovně.

```
getArrayNumber([#Data1], [#Data2], [#Data3]);
```

Kontext takové by potom měl vracet hodnoty odpovídající nadefinovaným stavům.

Pokud je třeba kontrolovat u widgetu nějakou jeho hodnotu v rámci nějakých mezí, je vhodné využít třetí typ stavů.

### 3.6.3. Treshold state

Tento typ bude mít stejně jako v předchozím případě pouze jednu vstupní proměnnou. Jeho stavy potom budou umožňovat porovnání hodnoty ze vstupní proměnné a z hodnoty z konkrétního stavu. Stav, který bude logicky pravdivý, v případě více stavů, pak ten, který má nejbližší hodnotu, bude vyhodnocen stejně jako v předchozím případě.

## 3.7. Události widgetu

U každého widgetu bude možnost nadefinování akce, která se má vykonat. Pokud se na widget klikne, provede se event mouse in, nebo mouse out. Každý widget proto bude mít tři vlastnosti, click, mouse in a mouse out. Obsah každé z těchto vlastností se po vykonání příslušné události zpracuje javascript funkcí eval.

## 3.8. Souborový systém

Pro pokročilejší tvorby widgetů budou potřeba například podkladové obrázky, svg schémata a podobně. Aplikace proto bude umožňovat nahrát do perzistentního úložiště jakýkoliv soubor, na který se potom může z celého systému odkazovat. Soubory budou nahrávány do stromové struktury s možností tvoření složek a podsložek. Pro přístup k souboru se potom bude používat následující url syntaxe.

```
/File/Cesta k souboru/Název souboru.přípona
```

## 3.9. Vlastnosti Editoru

V editoru bude uživatel editovat a tvořit nové stránky a widgety, a to tak, že bude upravovat jejich vlastnosti, nebo bude vkládat instance nových widgetů. Po kliknutí na instanci widgetu, se zobrazí kontextové menu s možností kopírování, smazání, přesunutí widgetu do jiné skupiny, nebo editace jeho vlastností, které jsou popsány v bodech výše. Při vybrání více instancí widgetů najednou, se zobrazí možnost zarovnání widgetů na stranu dle výběru. Widgety se budou pohybovat stylem drag & drop, nebo za pomoci šipek. Stejným způsobem půjde také měnit jejich velikost.

Každý z widgetů a stránek bude mít možnost konfigurace. U widgetů například, jak se bude jmenovat v panelu nástrojů, v jaké záložce bude umístěn, jestli bude možné po vložení jeho instance upravovat jeho velikost, nebo bude velikost fixní. Taktéž, jestli se s widgetem bude dát hýbat, otáčet s ním, nebo ho nějak jinak transponovat.

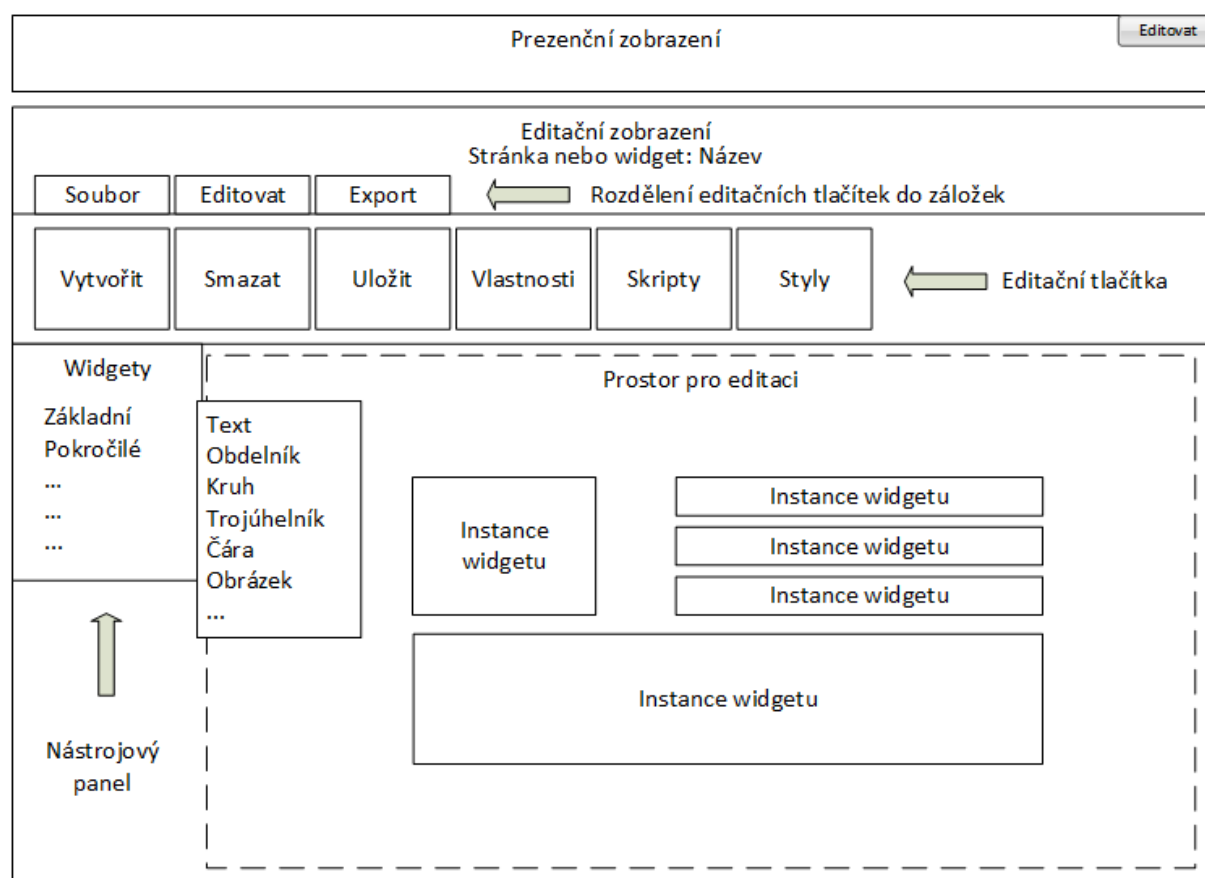
Editor umožní zobrazit seznam všech stránek a widgetů, taktéž seznam všech scriptů a stylů a umožní je zobrazit i editovat.

V editoru bude možné zobrazit průzkumník souborů, který umožní CRUD operace nad všemi soubory v systému.

Součástí editoru bude taky systém nápovědy. Všechny widgety a jejich vlastnosti budou mít vytvořenou nápovědu a ukázkou použití, aby bylo jednodušší s nimi pracovat. Nápovědy se budou ukládat ve formátu HTML stránky.

### 3.10. Design Editoru

Pro zobrazení editoru bude sloužit editační tlačítko, které se v případě dostatečných oprávnění zobrazí v pravém horním rohu obrazovky.



Obrázek 4 - Návrh editoru

V editoru potom v horní části bude informace o editované stránce nebo widgetu. Pod touto informací bude editační panel, v kterém budou různá editační tlačítka rozdělena do několika skupin. V levé části

potom bude dvouúrovňový panel nástrojů, který zobrazí a umožní přidávat nové widgety do stránky. Zbývající prostor bude sloužit k samotné editaci.

Pro editaci jednotlivých vlastností widgetu a stránky bude sloužit editační dialog, který bude mít v horní části název editovaného widgetu. Pod tímto názvem budou záložky, které jednotlivé vlastnosti rozdělují do příslušných skupin.

V řádcích budou poté zobrazeny jednotlivé vlastnosti. Název, editovaný výraz, výchozí hodnota, který se zobrazí, pokud nedorazí data ze serveru, zaškrťovací tlačítko, které říká, jestli se má výraz před zobrazením vyhodnotit Javascriptem. Poslední je ikona nápovědy, která zobrazí nápovědu pro jednotlivé vlastnosti widgetu.

Vlastnosti	Události	Stavy	Zabezpečení		OK	Storno																																
Název editovaného widgetu					Nápověda (?)																																	
<div> <u>Obecné</u> </div> <table> <thead> <tr> <th></th> <th>Výraz</th> <th>Výchozí hodnota</th> <th></th> </tr> </thead> <tbody> <tr> <td>Třída Widgetu</td> <td><input type="text"/></td> <td><input type="text"/></td> <td><input type="checkbox"/> ?</td> </tr> <tr> <td>Viditelné</td> <td><input type="text"/></td> <td><input type="text"/></td> <td><input type="checkbox"/> ?</td> </tr> <tr> <td colspan="4">.....</td> </tr> <tr> <td colspan="4"><u>Lokace</u></td> </tr> <tr> <td>Výška</td> <td><input type="text"/></td> <td><input type="text"/></td> <td><input type="checkbox"/> ?</td> </tr> <tr> <td>Šířka</td> <td><input type="text"/></td> <td><input type="text"/></td> <td><input type="checkbox"/> ?</td> </tr> <tr> <td colspan="4">.....</td> </tr> </tbody> </table>								Výraz	Výchozí hodnota		Třída Widgetu	<input type="text"/>	<input type="text"/>	<input type="checkbox"/> ?	Viditelné	<input type="text"/>	<input type="text"/>	<input type="checkbox"/> ?	.....				<u>Lokace</u>				Výška	<input type="text"/>	<input type="text"/>	<input type="checkbox"/> ?	Šířka	<input type="text"/>	<input type="text"/>	<input type="checkbox"/> ?	.....			
	Výraz	Výchozí hodnota																																				
Třída Widgetu	<input type="text"/>	<input type="text"/>	<input type="checkbox"/> ?																																			
Viditelné	<input type="text"/>	<input type="text"/>	<input type="checkbox"/> ?																																			
.....																																						
<u>Lokace</u>																																						
Výška	<input type="text"/>	<input type="text"/>	<input type="checkbox"/> ?																																			
Šířka	<input type="text"/>	<input type="text"/>	<input type="checkbox"/> ?																																			
.....																																						

Obrázek 5 - Návrh editoru vlastností

### 3.11. Ukládání dat

Data se budou ukládat do databáze. Pro persistenci souborů se použije ORM framework XPO firmy DevExpress [7].

Pro uložení stránky a widgetů se použije tabulka ContentTemplate. Tato tabulka bude mít následující sloupce.

- Name – název stránky nebo widgetu
- TemplateType – informace, jestli se jedná o stránku nebo widget
- SettingsXML – celé nastavení widgetu serializované do xml
- PropertiesXML – výchozí nastavení všech vlastností, které bude mít instance widgetu po vložení do stránky
- WidgetInstancesXML – instance widgetů, které jsou umístěné ve stránce nebo ve widgetu, plus všechny jejich nadefinované vlastnosti

Scripty a styly se budou ukládat do tabulky CustomContent. Tabulka bude mít tyto sloupce.

- Name – název skriptu nebo stylu
- Category – kategorie
- ContentType – informace, jestli se jedná o skript, nebo styl
- Content – samotná hodnota, buď skriptu, nebo stylu

### 3.12. Základní widgety

Základní widgety v aplikaci rozdělíme do několika skupin. První skupinou budou základní widgety. Tyto widgety budou zobrazovat nějaké základní informace a design.

**Text** – tento widget bude sloužit pro zobrazení textu. Jeho vlastnosti budou text, vnitřní a vnější okraje, vertikální a horizontální centrování, barva textu, velikost textu a dekorace textu.

**RichText** – jedná se o Text widget s podporou HTML ve vlastnosti text.

**DateTime** – téměř identický s text widgetem. Jeho výstup bude aktuální klientský čas. Jako vlastnost bude mít navíc formát výstupního času.

Geometrické widgety **Rectangle**, **Triangle**, **Circle**, **Horizontal Line** a **Vertical line** – tyto widgety budou mít následující vlastnosti stejné. Barva pozadí, barva čar, šířka čar, styl čar. Rectangle potom bude mít navíc radius zaoblení, minimální a maximální hodnotu pro vyplnění, směr vyplnění, barvu a aktuální hodnotu vyplnění. Triangle bude mít navíc akorát vlastnost typ, která říká, jestli je pravouhlý, nebo rovnostranný.

**Image** a **SVG** – tyto dva widgety budou zobrazovat obrázky, nebo svg dokumenty ze zadané url adresy.

Widgety pracující s uživatelským vstupem potom můžeme zařadit do druhé skupiny.

**Checkbox**, **Input**, **Textarea**, **Listbox** – tyto widgety budou očekávat nějaký vstup od uživatele, ať už vyplnění nějaké hodnoty, zaškrtnutí checkboxu, nebo vybrání ze seznamu. Všechny tyto widgety budou mít jednotnou vlastnost, která se bude jmenovat proměnná. Do této vlastnosti se bude očekávat následující syntaxe.

{#NázevProměnné}

Systém následně všechny vstupy z widgetu propíše do této kontrolky, taktéž zde bude fungovat oboustranný databinding. To znamená, že když se do přiřazené proměnné zapíše hodnota z jiného zdroje, tak se hodnota ve widgetu taky aktualizuje.

**Keypad** – pro zadání dat z dotykové obrazovky se využije tento widget. Widget bude reprezentovat numerickou klávesnici s desetinným oddělovačem pro zadávání hodnot. Pro potřeby systému je dostačující pouze numerická klávesnice. Ze zkušeností není dobré nechat uživatele psát vlastní texty. S takovými daty se pak špatně pracuje. Například stejný důvod poruchy by mohl uživatel zapsat různými způsoby, tak je lepší, nechat ho raději vybrat z předem připraveného seznamu. Widget stejně jako v předchozím případě bude využívat bindování na proměnnou.

**Repeater** – v některých případech bude potřeba zobrazit proměnlivé množství widgetů na základě dat ze serveru. V takovém případě bude třeba vytvořit widget repeater. Tento widget bude umět zobrazit widget jednoho typu v několika instancích. Počet instancí bude záviset na množství položek v proměnné

datasource. Pro každou položku ve zdroji dat poté proběhne skript, který se bude zadávat v poslední proměnné IterationScript. Do iteračního skriptu poté bude vstupovat tyto parametry:

- Item – aktuální položka z datasource
- Index – pořadí položky
- Count – celkový počet položek
- Datasource – reference na celý datasource

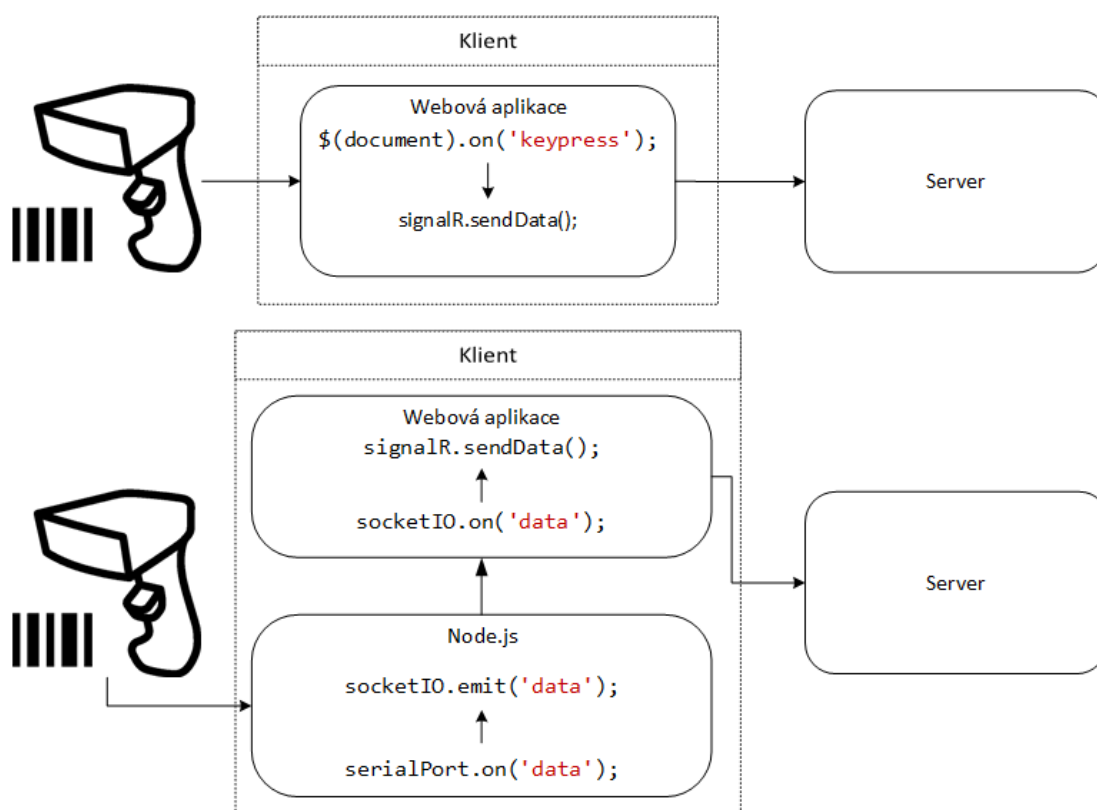
**IFrame** – widget typu iframe bude sloužit pro zobrazení jiného webu v kontextu stránky nebo widgetu. Tento widget bude mít vlastnost url, z které se vnořený web načte.

**HTML** – v případě, že ani jeden z výše sepsaných widgetů nebude postačovat, lze využít widget HTML, do kterého lze psát jakékoliv validní HTML a vytvořit si tak widget na míru.

### 3.13. Komunikace s periferiemi

Periferiemi budeme nazývat různé typy čteček čárových kódů, čtečky rfid karet, váhy a další zařízení. Pro získání dat z těchto zařízení máme dvě hlavní možnosti.

První je, že periferie bude pracovat v režimu emulace klávesnice. V takovém případě stačí odchyťovat přijaté znaky na úrovni webového prohlížeče. Pokud použijeme tento postup, nemusíme na klientskou stanici instalovat žádný podpůrný program, který by nám komunikaci zajišťoval. Výhodou tohoto řešení je, že se nemusí udržovat žádná další komponenta. Nevýhodou této komunikace je mírné zpoždění a pouze jednosměrná komunikace.



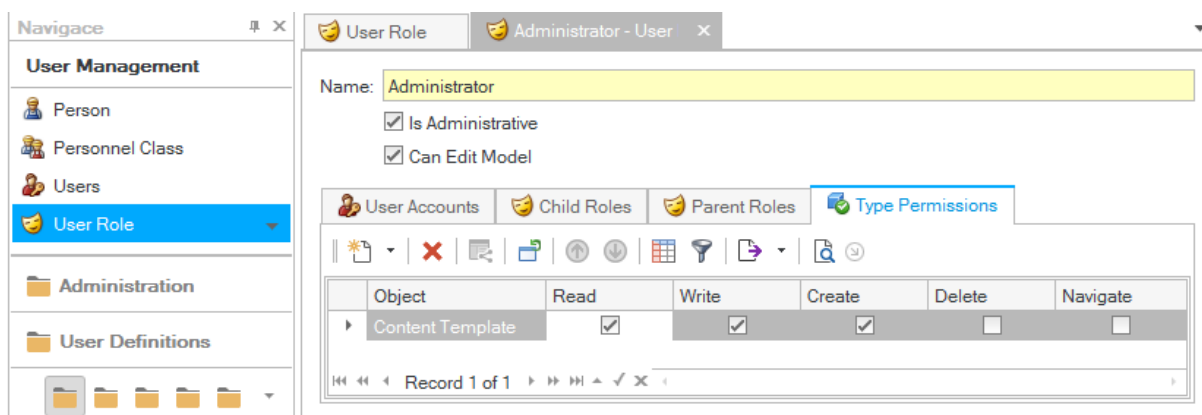
Obrázek 6 - Ukázka komunikace

Druhou možností je pro komunikaci využít například sériový port. V takovém případě musíme na klientskou stanici nainstalovat nějaký program, který tuto komunikaci zajistí. Vhodný kandidát je Node.js server se socket.io balíčkem. Socket.io balíček využijeme pro komunikaci klientské stanice a webovým prohlížečem, který na ní poběží. Výhodou řešení je rychlejší obousměrná komunikace. Nevýhodou této komunikace je, že se musí udržovat Node.js server na klientské stanici.

### 3.14. Security

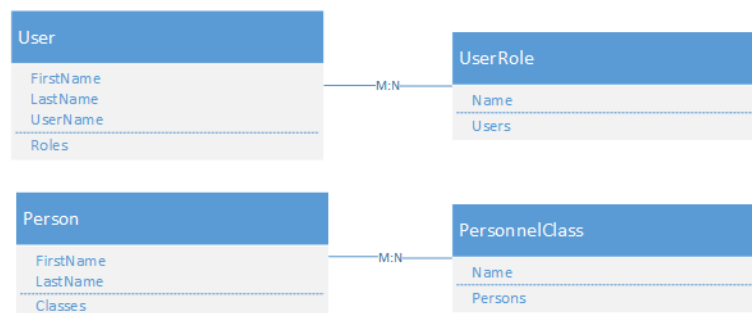
Jelikož bude aplikace nasazována v podnikových sítích, které mají nastavené doménové politiky, rozhodl jsem se, že pro ověření budu používat právě doménové ověřování. IIS nabízí pro toto ověření vestavěnou podporu a již v základním nastavení se může bezpečně používat. Jelikož vyvíjím webovou aplikaci s velkým množstvím javascriptu, je třeba v každé webové metodě kontrolovat, jestli se jedná o ověřený a zabezpečený požadavek. V klientské části, právě díky použití javascriptu a nemožnosti zabránit klientské editaci, se zabezpečení moc řešit nedá.

Zabezpečení aplikace bude aktivní pouze v případě, že poběží jako modul systému ImproveIT. V systému ImproveIT jsou definovány dvě úrovně zabezpečení. První úroveň zajišťuje zabezpečení samotné webové aplikace. Pro toto zabezpečení se bude využívat doménové ověření a podle přiděleného oprávnění se stránka nebo widget zobrazí v prezenčním režimu, v případě vyššího oprávnění se zobrazí editační tlačítko a umožní se tím editace stránky. Pro zajištění a definici oprávnění se využije security systém frameworku XAF. Pro tento typ security bude systém využívat typy User a UserRole.

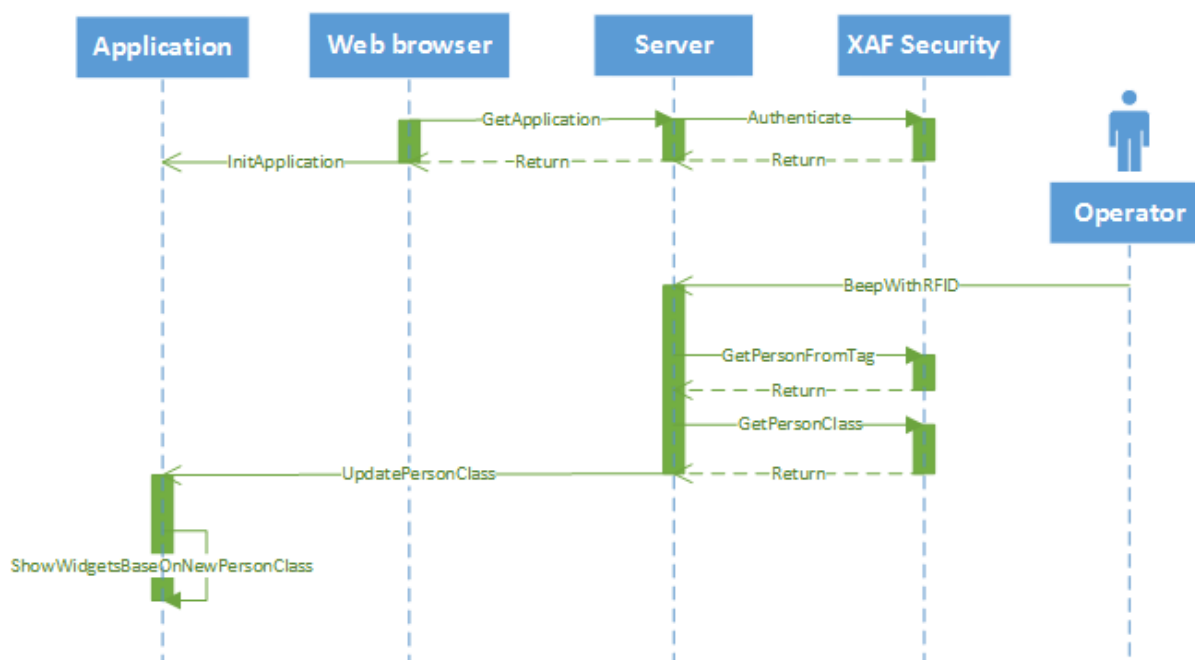


Obrázek 7 - Ukázka XAF security

Druhá úroveň zabezpečení bude na úrovni logiky aplikace na klientu a bude umožňovat zobrazovat a skrývat tlačítka v aplikaci a povolovat a zakazovat různé události na tlačítkách. Když proběhne přihlášení operátora, aplikace obdrží všechny role daného operátora a na základě těchto rolí upraví chování aplikace. Pro tento typ security bude systém využívat typy Person a PersonnelClass.



Obrázek 8 - Security třídy

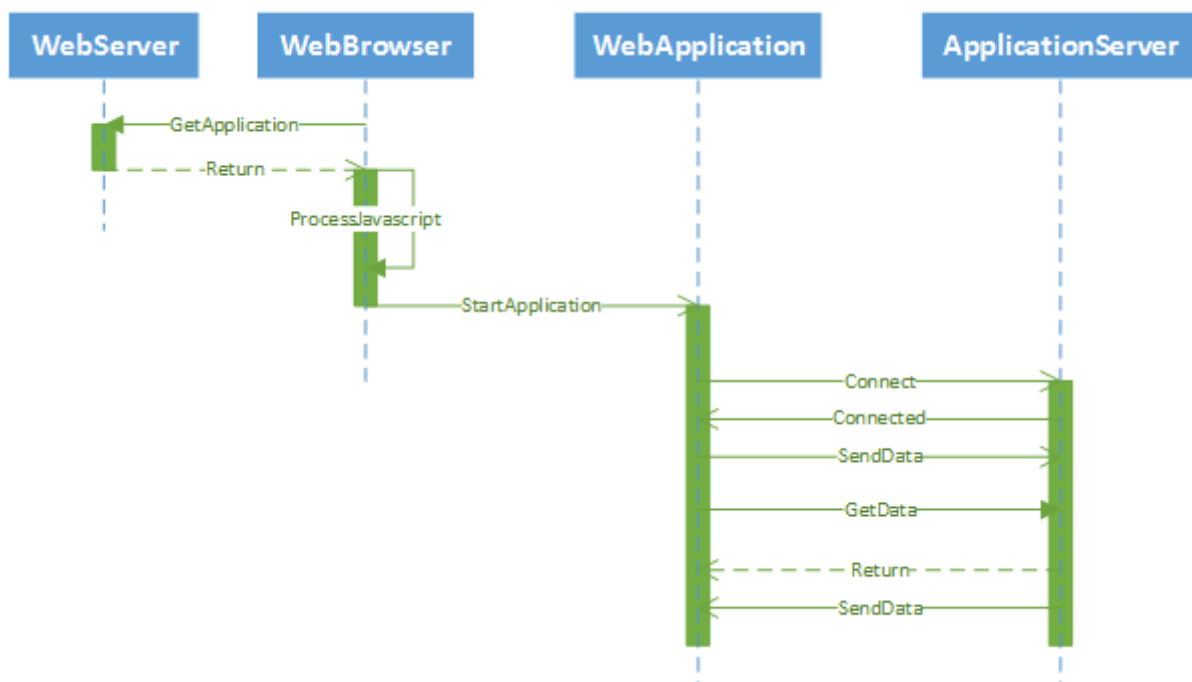


Obrázek 9 - Ukázka security

### 3.15. Inicializace komunikace

Po stažení HTML stránky a všech javascriptů se spustí a ziniculuje samotná aplikace. Po spuštění se aplikace připojí na server pomocí technologie SignalR [8]. Následně aplikace bude umožňovat komunikovat se serverem. Zprávy z klienta mohou být jak synchronní, tak asynchronní. Server umožňuje zasílat na klienta pouze zprávy asynchronní.





Obrázek 10 - Návrh komunikace

## 4. Testování SignalR

Technologie SignalR je pro aplikaci klíčovým prvkem. Tato technologie umožňuje téměř veškerou komunikaci mezi aplikací a aplikačním serverem. Proto je třeba tuto technologii, zejména komunikaci pomocí technologie websockets, otestovat. Na jeden server budou běžně připojeny desítky terminálů. V některých případech bude i jeden fyzický terminál využívat několika spojení na server.

Pro testování jsem vytvořil dvě konzolové aplikace. První aplikace slouží jako generátor klientů. Druhá funguje jako jednoduchý server. Pro sledování množství přenesených dat jsem využil wireshark [9].

### 4.1. Počet připojení

Pro scénář maximálního počtu připojení budeme počítat 100 terminálů. Tento počet odpovídá dílně, nebo továrně, která by měla přes 300 strojů. Každý terminál bude mít v průměru připojené 3 stroje. Celkově tedy budeme počítat 4 připojení na jeden terminál (3 stroje + kořenová obrazovka). Výsledný počet připojení tedy bude 400.

Každý klient si co 10 sekund (základní nastavení) udržuje aktivní připojení. Tato komunikace se skládá z dotazu a odpovědi. Velikost dotazu je 58 byte, velikost odpovědi 56 byte. Celkově jedna keep alive komunikace vyžaduje 114 byte. Při celkovém počtu 400 připojení budeme konstantě potřebovat 44.6 kb každých 10 sekund. Výsledná průměrná zátěž sítě by měla být přibližně 4.5 kb/s.

No.	Time	Source	Destination	Protocol	Length	Info
93	3.703621	192.168.1.1	192.168.1.10	TCP	56	5289 → 12345 [ACK] Seq=511 Ack=232 Win=66304 Len=0
102	4.993067	192.168.1.10	192.168.1.1	WebSocket	58	WebSocket Text [FIN]
111	5.047216	192.168.1.1	192.168.1.10	TCP	56	5288 → 12345 [ACK] Seq=550 Ack=282 Win=66304 Len=0
208	15.010002	192.168.1.10	192.168.1.1	WebSocket	58	WebSocket Text [FIN]
219	15.069577	192.168.1.1	192.168.1.10	TCP	56	5288 → 12345 [ACK] Seq=550 Ack=286 Win=66304 Len=0
292	25.030855	192.168.1.10	192.168.1.1	WebSocket	58	WebSocket Text [FIN]
302	25.084771	192.168.1.1	192.168.1.10	TCP	56	5288 → 12345 [ACK] Seq=550 Ack=290 Win=66304 Len=0
378	35.039935	192.168.1.10	192.168.1.1	WebSocket	58	WebSocket Text [FIN]
381	35.093609	192.168.1.1	192.168.1.10	TCP	56	5288 → 12345 [ACK] Seq=550 Ack=294 Win=66304 Len=0
515	45.056158	192.168.1.10	192.168.1.1	WebSocket	58	WebSocket Text [FIN]
519	45.119851	192.168.1.1	192.168.1.10	TCP	56	5288 → 12345 [ACK] Seq=550 Ack=298 Win=66304 Len=0
620	55.075291	192.168.1.10	192.168.1.1	WebSocket	58	WebSocket Text [FIN]
625	55.141992	192.168.1.1	192.168.1.10	TCP	56	5288 → 12345 [ACK] Seq=550 Ack=302 Win=66304 Len=0
748	65.112383	192.168.1.10	192.168.1.1	WebSocket	58	WebSocket Text [FIN]

Obrázek 11 - Ukázka keep alive komunikace

### 4.2. Množství dat

Pro množství přenesených dat budeme opět pracovat s počtem 100 terminálů. Na všechny tyto terminály musíme neustále posílat aktualizací data. Také musíme očekávat, že terminály někdo ovládá a vykonává na nich nějaké akce.

#### Data z klienta

Budeme předpokládat, že operátor nestojí u terminálu pořád, ale ve většině času se věnuje výrobě. U terminálu pak vykoná několik potřebných akcí a zase se věnuje výrobě. I tak počítejme, že operátor na každém terminálu vykoná co 5 sekund akci, která odešle data na server. Jako data si představme soubor formátu JSON. Pro data odeslaná z klienta na server počítejme velikost 300 byte. Tato velikost je dostačující pro jakékoliv data z klienta. I když je na jednom terminálu více strojů, v reálném čase může operátor ovládat vždy pouze jeden. Takže pro směr dat z klienta na server budeme počítat s pouze 100

klienty. Pro tento směr dat budeme využívat přibližně 300 byte \* 100 klientů / každých 5 sekund = 6 kb/s. Každá zpráva z klienta na server obsahuje také odpověď. Budeme předpokládat, že v odpovědi nejsou žádná extra data, takže její velikost bude konstantních 65 byte. Klient následně posílá ACK zprávu o velikosti 56 byte. Celkově se tedy dostáváme na číslo 8.4 kb/s.

No.	Time	Source	Destination	Protocol	Length	Info
393	25.841382	192.168.1.1	192.168.1.10	WebSocket	275	WebSocket Text [FIN] [MASKED]
394	25.842109	192.168.1.10	192.168.1.1	WebSocket	65	WebSocket Text [FIN]
397	25.902665	192.168.1.1	192.168.1.10	TCP	56	10499 → 12345 [ACK] Seq=1426 Ack=324 Win=66304 Len=0
464	29.857315	192.168.1.1	192.168.1.10	WebSocket	275	WebSocket Text [FIN] [MASKED]
465	29.858168	192.168.1.10	192.168.1.1	WebSocket	65	WebSocket Text [FIN]
468	29.917597	192.168.1.1	192.168.1.10	TCP	56	10499 → 12345 [ACK] Seq=1647 Ack=335 Win=66304 Len=0
543	33.719452	192.168.1.10	192.168.1.1	WebSocket	58	WebSocket Text [FIN]
549	33.783615	192.168.1.1	192.168.1.10	TCP	56	10499 → 12345 [ACK] Seq=1647 Ack=339 Win=66304 Len=0
551	33.862432	192.168.1.1	192.168.1.10	WebSocket	275	WebSocket Text [FIN] [MASKED]
552	33.863191	192.168.1.10	192.168.1.1	WebSocket	65	WebSocket Text [FIN]
555	33.924424	192.168.1.1	192.168.1.10	TCP	56	10499 → 12345 [ACK] Seq=1868 Ack=350 Win=66304 Len=0
603	37.866715	192.168.1.1	192.168.1.10	WebSocket	275	WebSocket Text [FIN] [MASKED]
604	37.867493	192.168.1.10	192.168.1.1	WebSocket	65	WebSocket Text [FIN]
607	37.927216	192.168.1.1	192.168.1.10	TCP	56	10499 → 12345 [ACK] Seq=2089 Ack=361 Win=66304 Len=0

Obrázek 12 - Ukázka komunikace z klienta na server

## Data ze serveru

Množství dat ze serveru bude rozhodně větší než v předchozím případě. Budeme počítat, že na všech 300 strojů, připojení budou proudit každou sekundu data s aktuálními hodnotami. Velikost dat bude jako v předchozím případě přibližně 300 byte. Při posílání dat ze serveru na klienta SignalR neočekává žádnou návratovou hodnotu. Klient tedy posílá pouze ACK zprávu. Při 300 strojích \* (300 byte + 56 byte) se dostáváme na hodnotu 106.8 kb/s. Kromě aktualizace aktuálních hodnot je třeba na klienta sem tam poslat větší aktualizaci stavů. Například při načtení nové zakázky, aktualizace přihlášených osob a jiné. Tyto operace se nedějí moc často. Pro simulační testy budeme počítat každých 60 sekund. Velikost zprávy bude v tomto případě větší. Budeme počítat 100 kb na zprávu. Tyto zprávy jsou podmíněny nějakou akcí operátora, takže budeme předpokládat jenom 100 připojení na fyzické terminály. Při (100 kb + 56 byte) \* 100 terminálů / každých 60 sekund se dostaneme na 166.7 kb/s.

No.	Time	Source	Destination	Protocol	Length	Info
15	4.001203	192.168.1.10	192.168.1.1	TCP	300	12345 → 11344 [PSH, ACK] Seq=251 Ack=1 Win=260 Len=246
18	4.056719	192.168.1.1	192.168.1.10	TCP	56	11344 → 12345 [ACK] Seq=1 Ack=497 Win=255 Len=0
43	8.001729	192.168.1.10	192.168.1.1	TCP	300	12345 → 11344 [PSH, ACK] Seq=497 Ack=1 Win=260 Len=246
46	8.055247	192.168.1.1	192.168.1.10	TCP	56	11344 → 12345 [ACK] Seq=1 Ack=743 Win=260 Len=0
69	10.097171	192.168.1.10	192.168.1.1	TCP	58	12345 → 11344 [PSH, ACK] Seq=743 Ack=1 Win=260 Len=4
72	10.152579	192.168.1.1	192.168.1.10	TCP	56	11344 → 12345 [ACK] Seq=1 Ack=747 Win=260 Len=0
116	12.002162	192.168.1.10	192.168.1.1	TCP	300	12345 → 11344 [PSH, ACK] Seq=747 Ack=1 Win=260 Len=246
119	12.057771	192.168.1.1	192.168.1.10	TCP	56	11344 → 12345 [ACK] Seq=1 Ack=993 Win=259 Len=0
168	16.003185	192.168.1.10	192.168.1.1	TCP	300	12345 → 11344 [PSH, ACK] Seq=993 Ack=1 Win=260 Len=246
171	16.057189	192.168.1.1	192.168.1.10	TCP	56	11344 → 12345 [ACK] Seq=1 Ack=1239 Win=258 Len=0
250	20.004233	192.168.1.10	192.168.1.1	TCP	300	12345 → 11344 [PSH, ACK] Seq=1239 Ack=1 Win=260 Len=246
253	20.058169	192.168.1.1	192.168.1.10	TCP	56	11344 → 12345 [ACK] Seq=1 Ack=1485 Win=257 Len=0
254	20.114517	192.168.1.10	192.168.1.1	TCP	58	12345 → 11344 [PSH, ACK] Seq=1485 Ack=1 Win=260 Len=4
258	20.168291	192.168.1.1	192.168.1.10	TCP	56	11344 → 12345 [ACK] Seq=1 Ack=1489 Win=257 Len=0

Obrázek 13 - Ukázka komunikace ze serveru

Při sečtení veškerých hodnot komunikace se dostáváme na necelých 300 kb/s. Toto využití sítě reprezentuje zapojení 300 strojů na 100 terminálech s téměř neustálou interakcí operátorů s každým terminálem. I kdyby taková situace nastala, výsledná hodnota je zanedbatelná s celkovou kapacitou gigabitových podnikových sítí.

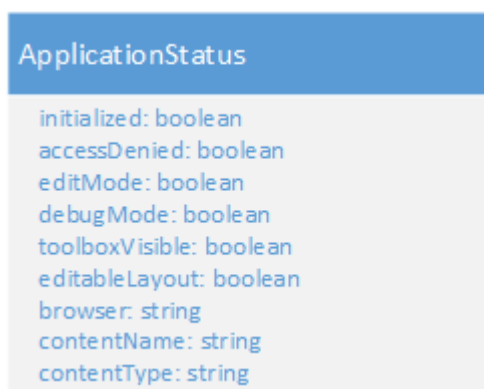
## 5. Implementace

### 5.1. Služby

Pro rozdělení logiky, jsem se rozhodl využít možnosti implementace angular [10] služeb. Tyto služby jsou vlastně singleton instance, které obstarávají různé funkcionality systému. Služby jsou díky dependency injection [11] dostupné ze všech kódů angular aplikace. V následujících odstavcích popíšu funkce jednotlivých služeb.

#### 5.1.1. ApplicationStatus

Tato služba drží informace o stavech aplikace.



Obrázek 14 – ApplicationStatus

**Initialized** – Informace o úspěšném načtení aplikace. Příznak se nastaví po úspěšném dokončení prvního angular cyklu.

**AccessDenied** – V případě, že se uživatel pokusí načíst stránku nebo widget, na který nemá oprávnění, nastaví se tento příznak. V kořenovém body elementu je umístěný speciální widget, který reaguje na tento příznak. Po nastavení tohoto příznaku se přes celou obrazovku zobrazí chybová hláška "Přístup odepřen".

---

```
<forbidden-screen-widget style="position: fixed; top: 0px; width: 100%; height: 100%;  
z-index: 10001" ng-show="status.accessDenied"></forbidden-screen-widget>
```

---

Zdrojový kód 1 - Ukázka widgetu Přístup odepřen

**EditMode** – Globální příznak, pomocí kterého se zapíná a vypíná editační mód.

**DebugMode** – Příznak, který řídí zobrazování dat ze serveru. V testovacích případech lze nastavit, aby byla data zobrazována z testovací lokální kolekce. V takovém případě se musí nastavit tento příznak.

**ToolboxVisible** – Uživatelské rozhraní editačního režimu umožňuje zobrazování a skrývání panelu nástrojů. Tato funkcionality je řízena tímto příznakem.

**EditableLayout** – Ne vždy je na základě oprávnění umožněna editace. Pokud má být zobrazeno editační tlačítko, musí se nastavit tento příznak.

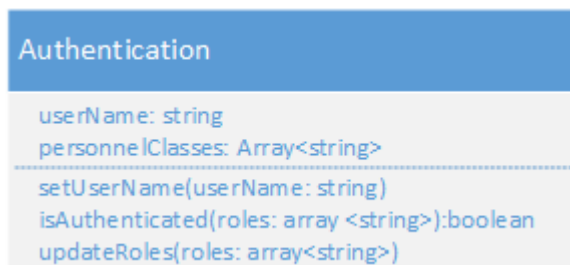
**Browser** – Informace o prohlížeči, ve kterém je aktuálně aplikace spuštěna.

**ContentName** – Název aktuálně editovaného widgetu nebo stránky.

**ContentType** – Informace, jestli se edituje widget, nebo stránka.

### 5.1.2. Authentication

O Informace o aktuálně přihlášených rolích operátora se stará tato služba.



Obrázek 15 – Authentication

**UserName** – Aktuálně přihlášený uživatel.

**PersonnelClasses** – Informace o rolích všech přihlášených operátorů. Na základě těchto rolí se aktualizuje zobrazení všech widgetů.

**SetUserName** – Po inicializaci aplikace se zavolá tato metoda a předá se službě informace o aktuálně přihlášeném uživateli.

**IsAuthenticated** – Metoda, která zjistí, jestli je alespoň jedna přihlášená role shodná s rolemi v parametru funkce.

**UpdateRoles** – Po uživatelském vstupu, například přiložení RFID karty, server pošle aktualizovanou kolekci aktuálně přihlášených rolí.

### 5.1.3. ContextMenu

Tato služba se stará o zobrazování a vykonávání akcí kontextového menu.



Obrázek 16 – ContextMenu

**Show** – Metoda pro zobrazení kontextového menu. Jako parametr přímá objekt, který implementuje `showContextMenuOptions` rozhraní.

---

```
interface showContextMenuOption{
    items: contextMenuItem;
}
interface contextMenuItem {
    [category: string]: {
        [item: string]: {
            text: string;
            click?: Function;
            items?: contextMenuItem;
        }
    }
}
```

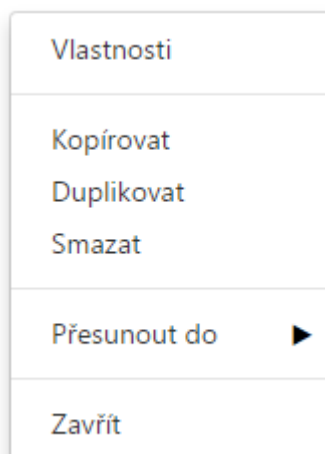
```

    };
};
}

```

*Zdrojový kód 2 – ShowContextMenuOption*

Před zobrazením se zpracuje options objekt a pro každý menu item se vygeneruje příslušné HTML. Design kontextového menu vychází z knihovny bootstrap [12].



*Obrázek 17 - Ukázka contextového menu*

#### 5.1.4. CustomContent

Tato služba spravuje veškeré uživatelské skripty a styly. Instance služby si drží v paměti všechny uživatelské styly a skripty a v případě potřeby je integruje do webového prohlížeče.

Při editaci skriptů a stylů se snažím nově vytvořené metody a styly hned integrovat do prohlížeče. Problém je, že prohlížeč si pamatuje všechny předchozí verze skriptů a stylů, takže v případě editace a tvorby nové stránky nebo widgetu je někdy třeba stránku aktualizovat, aby došlo k úplnému restartu aplikace a byly načteny jenom poslední verze stylů a skriptů.

CustomContent	CustomContentModel
<pre> scripts: dictionary&lt;string, CustomContentModel&gt; styles: dictionary&lt;string, CustomContentModel&gt; ..... registerScripts(scripts: array&lt;CustomContentModel&gt;) registerStyles(styles: array&lt;CustomContentModel&gt;) deleteContent(content: CustomContentModel) setContent(content: CustomContentModel) getContent(name: string, type: string) </pre>	<pre> contentType: string content: string name: string category: string </pre>

*Obrázek 18 – CustomContent*

**Scripts a styles** – Jedná se o kolekce, které reprezentují všechny styly a skripty.

**RegisterScripts a RegisterStyles** – Po inicializaci aplikace je třeba předat službě kolekci všech stylů a skriptů. Pro tuto akci slouží tyto metody.

**SetContent, DeleteContent, GetContent** – Metody pro základní operace se skripty a styly. Tyto metody udržují konzistenci mezi lokální pamětí všech stylů a scriptů a persistentním stavem na serveru.

#### 5.1.5. DataLayer

Tato služba zajišťuje práci se všemi serverovými proměnnými.

Pro implementaci této služby bylo třeba vytvořit tři důležité objekty.

**DataFromServer** – Jedná se o lokální obraz posledních hodnot všech dat, která dorazila ze serveru.

**Requests** – Mapovací objekt, který propojuje jednotlivá data s konkrétní proprietou, v které jsou data požadována.

**RequestsToReplace** – Dočasný objekt, který si pamatuje všechny data, u kterých došlo ke změně od poslední aktualizace.



```
parseRequest(requestId: string)
setDebugValue(requestId: string, value: string)
getDebugValue(requestId: string)
getValue(requestId: string, debug?: boolean): Object
getStringValue(requestId: string, debug?: boolean): string
removeProperty(propertyId: number, requestName: string)
updateRequests(forceUpdate?: boolean)
clearRequestsToReplace()
```

Obrázek 19 – DataLayer

**ParseRequest** – Metoda pro zpracování serverové proměnné. Dotaz na server může obsahovat serverovou i lokální část. Tato metoda vstupní řetězec zpracuje a vrátí právě tyto dvě části.

**SetDebugValue, GetDebugValue** – V případě, že se aplikace přepne do ladícího režimu, je třeba všechny data ze serveru nahradit ladícími daty. Práci s těmito daty umožňují tyto metody.

**GetValue, GetStringValue** – Metody pro získání posledních dat ze serveru. Metoda GetValue vrátí takovou hodnotu, jaká přišla ze serveru. GetStringValue vrátí vždycky čitelný string. V případě, pokud se jedná o primitivní typ, zavolá se pouze metoda toString(). V ostatních případech se hodnota převede na string pomocí JSON.stringify().

**RemoveProperty** – Tato metoda odstraní referenci na již neexistující proprietu. Například při smazání widgetu.

**UpdateRequests** – Tato metoda zajistí propisání všech nových dat do příslušných propriet a jejich widgetů.

**ClearRequestsToReplace** – Po úspěšném zapsání všech změn v datech je třeba tuto kolekci vyčistit, aby se při přijatých datech mohla opět naplnit.

#### 5.1.6. Dialog

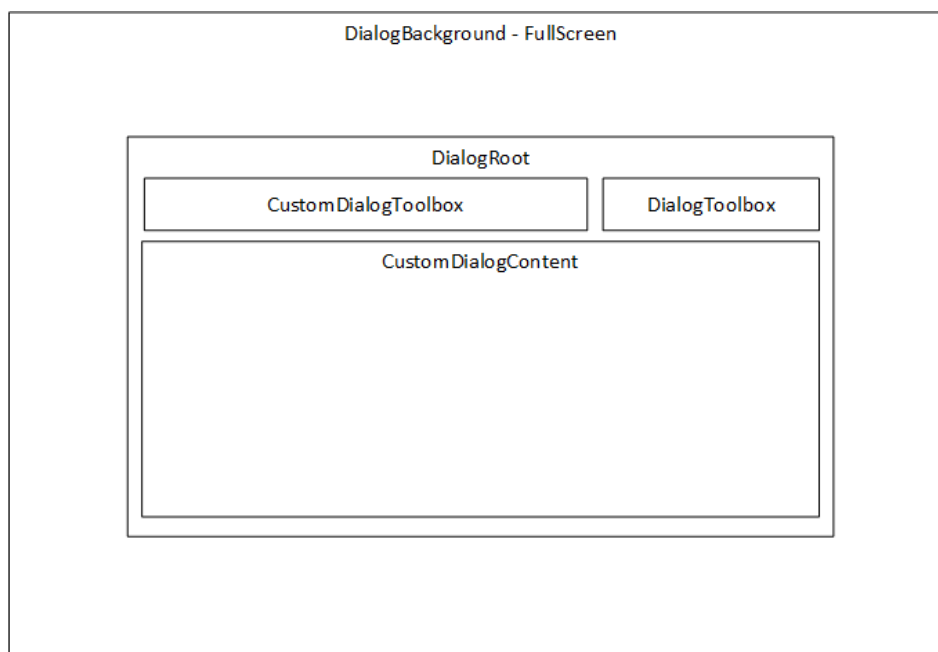
Tato služba zajišťuje zobrazování různých dialogů.

Dialog	ShowCustomModalOption
<code>toast(text: string, type: string, timeout: number)</code> <code>showCustomModal(options: showCustomModalOption)</code> <code>showTemplateList(templateType: string)</code> <code>showResourceDialog()</code> <code>showLanguageDialog()</code> <code>showDebugDialog()</code> <code>showFileDialog()</code> <code>showContentProperties()</code> <code>showCreateNew()</code> <code>showSaveAs()</code>	<code>directive: string</code> <code>size: Object</code> <code>acceptOnEnter?: boolean</code> <code>okButton?: boolean</code> <code>data?: any</code> <code>widget?: widget</code>

Obrázek 20 - Dialog, ShowCustomModalOption

**Toast** – Pro zobrazení rychlých zpráv na obrazovku, takzvaných "toastů" se používá tato metoda. Pro implementaci této metody jsem využil komponentu dxToast. V posledních týdnech jsme díky testům výkonu zjistili, že komponenta je zbytečně robustní a zobrazení zprávy na terminálech trvá přes 250ms. Na základě těchto výsledků byla komponenta odstraněna a nahrazena vlastním kódem. Nyní zobrazení zprávy trvá pod 10ms.

**ShowCustomModal** – Metoda se používá pro zobrazení všech dialogů v editačním rozhraní aplikace. Logika metody zajistí zobrazení základní kostry všech dialogů. Dialog se následně nastaví podle vstupních parametrů. Upraví se jeho velikost. Dialogům, které neobsahují textový editor, se nastaví zavření po zmáčknutí klávesy enter. Některé dialogy potřebují extra data pro zobrazení, pro tuto potřebu slouží objekt data.



Obrázek 21 - Schéma dialogu



Ostatní show metody jsou helpery postaveny nad metodou ShowCustomModal. Každá z těchto metod musí mít implementovanou vlastní angular direktivu, která reprezentuje CustomDialogContent a CustomDialogToolbox.

```
showContentProperties() {  
  this.showCustomModal({  
    directive: "property-dialog",  
    widget: service.mainContent,  
    size: {  
      width: 800,  
      height: 600  
    },  
    data: "mainContent",  
    okButton: true  
  });  
}
```

*Zdrojový kód 3 - Ukázka ShowCustomModal*

### 5.1.7. Editor

Tato služba obsahuje helpery pro práci s widgety v editačním zobrazení.



*Obrázek 22 – Editor*

**BorderSnapEnabled, GridSnapEnabled** – Tyto příznaky ovlivňují chování widgetu při změně pozice. Příznak BorderSnapEnabled zajišťuje, že se widget při přesunu vždy "přilepí" na sousedící widget. GridSnapEnabled nastavuje přesun widgetu vždy po mřížce. Umožňuje tak přesnější umístění widgetu.

**SelectionCaption** – Text, který se zobrazí při vybrání widgetu. Zobrazuje vždy vybraný typ widgetu, případně informaci, že je widgetů vybraných víc.

**Select a Deselect** – Metody zajišťují označení a odznačení widgetů, na které se kliklo. Označené widgety se ukládají do interní paměti této služby.

Metody **Move**, **Small**, **Large** pracují s vybranými widgety a upravují jejich pozici a velikost. Po zavolání příslušné metody se vždy rozměr změní o 1px.

---

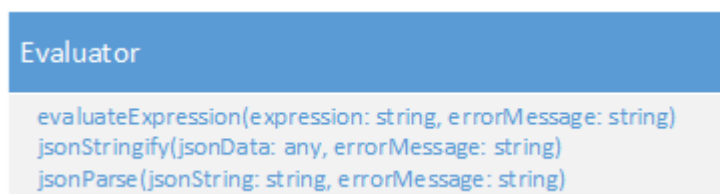
```
moveUp() {
  this.selectedWidgets.forEach((selectedWidget) => {
    selectedWidget.value.decrementPropertyValue("top");
  });
  service.webClient.angularApply();
}
```

---

*Zdrojový kód 4 - Ukázka metody MoveUp*

#### 5.1.8. Evaluator

Tato služba obaluje javascript funkci eval a odchyťává výjimky. Každá z metod má potom přednastavenou chybovou zprávu.



*Obrázek 23 – Evaluator*

#### 5.1.9. Hotkeys

Služba spravující klávesové zkratky. Pro zpracování zkratek jsem se rozhodl využít již hotový JQuery [13] plugin "hotkeys" [14]. Tento plugin umožňuje rozšířit událost bind o textový popis, na kterou klávesu má reagovat.

---

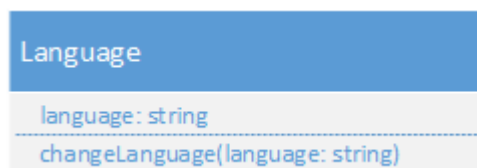
```
$document.bind('keydown', 'ctrl+s', () => {
  if (this.canProcessShortcut()) {
    service.transport.saveContent();
    return false;
  }
});
```

---

*Zdrojový kód 5 - Ukázka klávesové zkratky*

#### 5.1.10. Language

Tato služba obstarává lokalizaci aplikace. Pro překlad jsem se rozhodl využít již hotový modul, který obsahuje \$translate provider, který si drží v paměti překládací slovník. Součástí je i filtr, který zajišťuje již samotný překlad.



*Obrázek 24 – Language*

**Language** – Informace o aktuálním jazyku aplikace.

**ChangeLanguage** – Metoda, která využívá \$translate [15] provider a nastavuje nový jazyk. Jazyk je taktéž uložen do cookies prohlížeče a vždy po startu aplikace je z cookies načten. Výchozí jazyk je čeština.

---

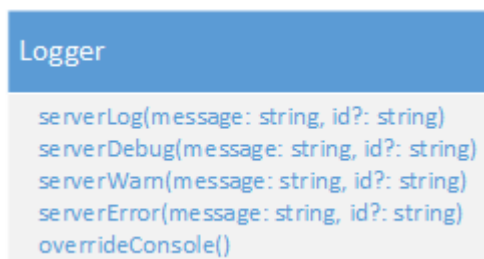
```
<div class="text">{{ 'EVENTS' | translate}}</div>
```

---

*Zdrojový kód 6 - Ukázka použití překladu*

#### 5.1.11. Logger

Služba obalující základní logování javascriptu a zajišťuje přeposílání logu na server, případně na klienta.



*Obrázek 25 - Logger*

V průběhu vývoje jsem zjistil, že nikdy není dost logů. Pro to jsem navrhnul systém logování, který logy pošle jak na server, tak je uloží i na terminálu. Hledání chyb je poté mnohem jednodušší.

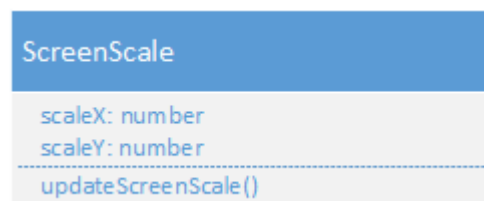
**ServerLog, ServerDebug, ServerWarn, ServerError** – Všechny tyto metody zalogují události jak do konzole prohlížeče, tak i v případě připojení SignalR klienta je odešlou na server.

**OverrideConsole** – Jelikož javascript umožňuje přepsání prototypu funkce, tak každá logovací metoda na objektu console je přepsána a zároveň posílá všechny logy přes socketIO na nodejs server, který běží na terminálu. Tyto logy jsou pak na terminálu uloženy a lze je zpětně dohledat.

#### 5.1.12. ScreenScale

Tato služba zajišťuje změnu velikosti stránky podle rozlišení klienta.

Každá stránka má ve svém nastavení uložena tři základní nastavení. Jestli celkově umožňuje změnu velikosti a jestli je změna pouze vertikální, nebo jenom horizontální. Po případě kombinace, která umožňuje proporcionální změnu velikosti, nebo deformační změnu.



*Obrázek 26 - ScreenScale*

**UpdateScreenScale** – Tato metoda přepočítá transformaci, která se aplikuje na webovou stránku, v závislosti na výše zmíněném nastavení.

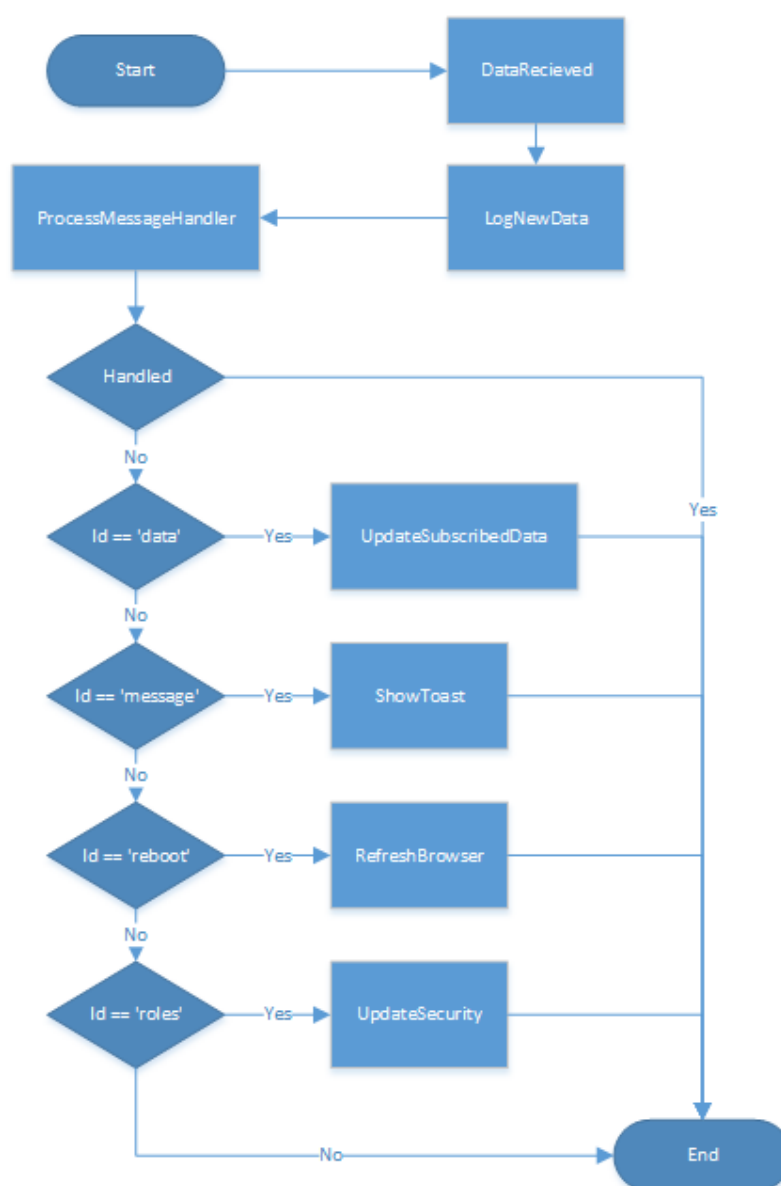
#### 5.1.13. SignalR

Tato služba obstarává veškerou komunikaci se serverem za pomoci SignalR.

SignalR	signalRConnectOption
<code>connect(option?: signalRConnectOption)</code> <code>subscribe(source: string, requests: array&lt;string&gt;)</code> <code>unsubscribe(source: string, requests: array&lt;string&gt;)</code> <code>processMessageOnServer(id: string, data: object)</code>	<code>urlAddress: string</code> <code>friendlyId?: string</code> <hr/> <code>connectedHandler?: Function</code> <code>processMessageHandler?: Function</code> <code>reconnectingHandler?: Function</code> <code>reconnectedHandler?: Function</code> <code>disconnectedHandler?: Function</code>

Obrázek 27 - SignalR

Každá aplikace může mít v konfiguračním souboru nastavenou adresu, na kterou se tato služba automaticky připojí. Pokud tato informace v konfiguraci chybí, je možné službu připojit na server ručně.



Obrázek 28 - SignalR DataReceived

**Connect** – Tato metoda inicializuje SignalR klienta a připojí ho na server. Pro připojení tato metoda přijímá objekt, který obsahuje url adresu a takzvané friendlyId, které identifikuje klienta na serveru. Konfigurační objekt dále obsahuje reference na callback metody. Tyto metody se volají vždy podle kontextu jejich názvu.

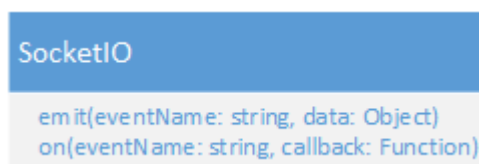
**Subscribe, Unsubscribe** – Komunikace se serverem funguje na principu publish – subscribe. Vždy když chce aplikace získávat nějaká data, měla by si o ně pomocí metody subscribe požádat.

**ProcessMessageOnServer** – Metoda pro odeslání dat na server.

Po obdržení nových dat ze serveru aplikace zavolá callback, který pokud se zpracuje, tak vykonávání metody končí. Pokud ne, tak se na základě identifikace obdržené zprávy rozhodne, co se s daty udělá.

#### 5.1.14. SocketIO

Tato služba zajišťuje komunikaci s lokálním hardwarem klienta. Po startu aplikace se automaticky připojuje na adresu localhost, kde očekává běžící nodejs server.



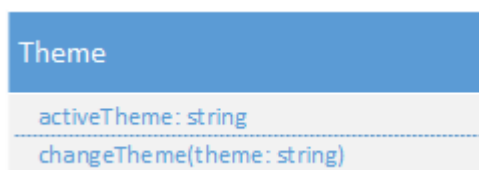
Obrázek 29 - SocketIO

**Emit** – Tato metoda odesílá zprávy na server.

**On** – Metoda slouží k registraci eventů. Jakmile server odešle zprávu, zpracuje se callbackem zadaným do této funkce.

#### 5.1.15. Theme

Služba zajišťující změnu designu editoru aplikace. Aktuálně nastavené téma aplikace je uloženo v cookies. Integrace a nastavení různých témat je zajištěno změnou kořenového CSS stylu. Pro vytvoření nového tématu stačí vytvořit příslušný CSS soubor a nakonfigurovat nové barvy.



Obrázek 30 - Theme

#### 5.1.16. Toolbox

Tato služba drží informace o všech widgetech a zobrazuje je v nástrojovém panelu. Služba si udržuje v paměti dvouúrovňové navigační menu, které se následně toolbox widgetem vykreslí v editačním zobrazení.

Toolbox	RegisterWidgetOption
<code>registerWidget(option: registerWidgetOption)</code> <code>registerWidgets(options: array&lt;registerWidgetOption&gt;)</code>	<code>name: string</code> <code>category: string</code> <code>directive: string</code>

Obrázek 31 - Toolbox

### 5.1.17. Transport

Tato služba obaluje get a post metody na webový server. Služba taktéž umožňuje automatické stahování různého obsahu z aplikace.

Transport
<code>saveContent(name?: string, type?: string)</code> <code>deleteContent(name: string, type: string)</code> <code>downloadFile(url: string)</code> <code>exportContent(name: string, type: string)</code> <code>exportContents(type: string)</code> <code>importContents(files: FileList)</code>

Obrázek 32 - Transport

**SaveContent** – Pokud metoda nedostane žádné parametry, tak uloží editovaný obsah stránky pod aktuálním názvem a typem. Také lze metodě zadat název a typ pod jakým má aktuální obsah uložit.

**DeleteContent** – Smazání stránky nebo widgetu. Po úspěšném smazání je zobrazen toast.

**DownloadFile** – Pro vynucení stáhnutí souboru se využívá tato metoda. Služba transport po inicializaci vytvoří neviditelný HTML objekt IFramu. Tato metoda propíše zadané url do url IFramu a tím vyvolá stažení souboru.

**Export a Import** – Metody pro importování a exportování obsahu. Celý obsah stránky nebo widgetu se serializuje do xml souboru. Metody podporují i hromadný import a export v zip formátu.

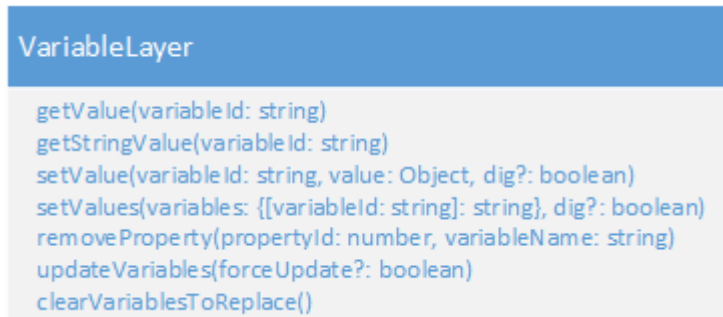
### 5.1.18. VariableLayer

Tato služba zajišťuje práci s globálními proměnnými. Služba pracuje na podobném principu jako již zmiňované služba DataLayer. Také zde jsou potřeba tři důležité objekty. Obě služby sdílejí stejnou logiku pro aktualizaci dat.

**VariableValues** – Do tohoto objektu se ukládají všechny globální proměnné.

**Variables** – Mapování jednotlivých proměnných na property, ve kterých jsou použity.

**VariablesToReplace** – Dočasný objekt, který si pamatuje všechny změněné property a následně provede jejich aktualizaci.



Obrázek 33 - VariableLayer

**GetValue** – Metoda pro získání globální proměnné.

**GetStringValue** – Metoda pro získání globální proměnné ve string formátu.

**SetValue, SetValues** – Metoda pro přidání nové proměnné do paměti. Posledním parametrem umožňuje vyvolat propsání nově změněných proměnných do zobrazení.

**RemoveProperty** – Metoda pro odstranění mapování. Například při smazání widgetu a tím pádem smazání i property, ve které mohla být reference na globální proměnnou.

**ClearVariablesToReplace** – Po každé aktualizaci je třeba vyčistit dočasnou kolekci všech změněných propert, aby se zde mohly zapsat nové změny.

**UpdateVariables** – Metoda způsobí zobrazení všech upravených propert.

#### 5.1.19. WebClient

Jedná se o službu, které obsahuje globální helpery. Služba si drží informace o posledním id widgetu a property. Stará se o spuštění aktualizace při změně nějakého stavu widgetu.



Obrázek 34 - WebClient

**GetBaseUrl** – Tato metoda vrátí kořenové url aplikace, pod kterým běží na IIS serveru. Vychází z HTML tagu "Base".

**GetCurrentUrl** – Vráti aktuální url adresu prohlížeče.

**GetNewPropertyId** – Inkrementuje a vrátí nové id property. Každá properta má své jedinečné id.

**GetNewWidgetId** – Inkrementuje a vrátí nové id widgetu. Každý widget má své jedinečné id.

**AngularApply** – Zkontroluje aktuální stav angular cyklu a pokud cyklus neběží, tak ho spustí.

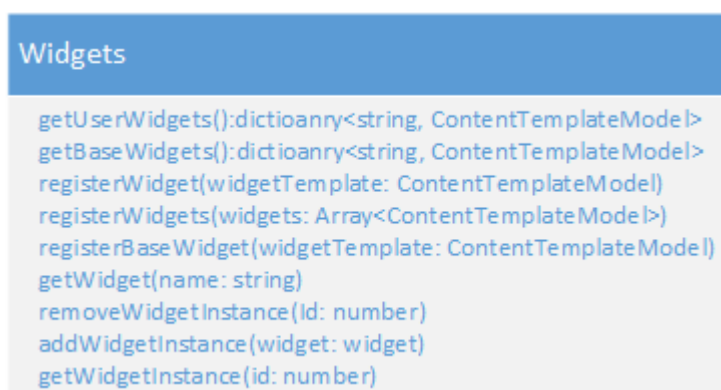
**UpdateVariables, UpdateRequests** – Reference na služby DataLayer a VariableLayer.

**UpdateStates** – Aktualizuje všechny stavy widgetů u kterých došlo v proprietách ke změně.

**BindLeaveConfirmation** – Nastaví prohlížeč, aby uživateli nedovolil odejít ze stránky v editačním režimu bez upozornění.

### 5.1.20. Widgets

Jak z návrhu vyplývá, musí mít aplikace implementovanou sadu základních widgetů. Pro snadnější implementaci jsem vytvořil službu widgets.



Obrázek 35 - Widgets

**GetUserWidgets** – Metoda vrátí modely všech uživatelsky vytvořených widgetů.

**GetBaseWidgets** – Metoda vrátí model všech systémových widgetů.

**RegisterWidget, RegisterWidgets** – Metody slouží pro registraci uživatelských widgetů při startu aplikace.

**RegisterBaseWidget** - Metoda jako své parametry přijímá šablonu widgetu. Tato šablona musí obsahovat tyto vlastnosti.

- Name – název widgetu
- Template – HTML šablona widgetu s podporou angular syntaxe.
- Properties – Vlastnosti, které bude mít nově vytvořený widget. Tyto vlastnosti se při registraci automaticky doplní o základní vlastnosti, které má každý widget.
- Settings – Základní nastavení widgetu. V tomto nastavení se uvádí, jestli může widget měnit velikost, pohybovat se, nebo otáčet.
- Link – Pro složitější implementace widgetu využijeme i vlastnost link, což je javascript callback volaný ve chvíli, kdy probíhá zpracovávání widgetu angularem.

Ukázka implementace widgetu typu HTML.

```
var template = `
<div class='html-content' style='width: 100%; height: 100%;' ng-bind-
html='widget.propertyGroups.properties.dictionary.html.calcValue | projectTranslate:
service.projectLanguage.language | toTrustedHTML'></div>
```



```

<div style= 'width: 100%; height: 100%; top: 0px; position: absolute' ng-
show='status.editMode'> </div>`;
toolbox.registerWidget({ category: "Advanced", name: "HTML", directive: "html" });
widgets.registerBaseWidget({
    name: "html",
    template: template,
    properties: [
        { name: "html", value: "<div style='width: 100%; height: 100%; border: 2px
solid white; box-sizing: border-box'></div>", group: "base", editor: "html", helpUrl:
"Help/Unique/html.html" }
    ],
    settings: {
        editor: {
            resize: {
                resizable: "true", horizontal: "true", vertical: "true"
            },
            drag: {
                draggable: "true"
            },
            transform: {
                rotation: "true", translation: "true"
            }
        }
    }
});

```

---

*Zdrojový kód 7 - Ukázka tvorby widgetu*

Jak můžeme v ukázce vidět, tak HTML widget má pouze jednu vlastnost, a to HTML v něm zobrazené. V proměnné template je potom HTML rozložení widgetu, s odkazem právě na tuto vlastnost. Direktiva NG-BIND-HTML nám zajistí, že se obsah z této vyrenderuje v samotném HTML. Hodnota v NG-BIND-HTML obsahuje ještě dva filtry, první zajišťuje překlad, druhá kontroluje strukturu HTML, jestli je v pořádku a nemůže nám nějak uškodit.

## 5.2. Model Aplikace

Pro ukládání dat jsem musel vymyslet model aplikace, který se celý dá serializovat například do xml, nebo jiného textového formátu. Tato serializace pak umožňuje jednoduché ukládání všech stránek a widgetů a také jejich import a export.

ContentTemplateModel reprezentuje model vytvořené stránky nebo widgetu. Každý z těchto objektů pak má kolekci svých vlastností a také kolekci instancí widgetů. Tento model také obsahuje nastavení widgetu a stránky.

WidgetInstanceModel reprezentuje instanci vloženého widgetu. Tento widget se musí odkazovat na nějaký ContentTemplateModel, který definuje, jak má instance vypadat. Každá z instancí má také kolekci svých vlastností. V těchto vlastnostech je například pozice a rozměry widgetů, různé události, nebo také nově vytvořené vlastnosti konkrétního widgetu.

PropertyModel je model nějaké konkrétní vlastnosti. Tento model kromě názvu a hodnoty obsahuje i další důležité informace. Například výchozí hodnotu, jestli se má výraz zpracovat, pořadí, url adresy na které se nachází nápověda. Také obsahuje kolekci dodatečných hodnot. Tato kolekce obsahuje vždy klíč a hodnotu.

```

classDiagram
    class ContentTemplateModel {
        name: string
        templateType: string
        widgetInstances: Array<WidgetInstanceModel>
        templateGroups: Array<TemplateGroupModel>
        settings: ContentSettingsModel
        properties: Array<PropertyModel>
    }
    class WidgetInstanceModel {
        id: number
        name: string
        groupId: number
        properties: Array<PropertyModel>
    }
    class TemplateGroupModel {
        id: number
        name: string
        order: number
    }
    class ContentSettingsModel {
        toolbox: ToolboxSettingsModel
        editor: EditorSettingsModel
    }
    class PropertyModel {
        name: string
        displayName: string
        value: string
        defaultValue: string
        group: string
        order: number
        evaluate: boolean
        editor: string
        datasource: Object
        helpUrl: string
        additionalValues: Array<AdditionalPropertyValueModel>
    }
    class ToolboxSettingsModel {
        category: string
        name: string
    }
    class EditorSettingsModel {
        resize: ResizeSettingsModel
        drag: DragSettingsModel
        transform: TransformSettingsModel
    }
    class AdditionalPropertyValueModel {
        name: string
        value: string
    }
    class ResizeSettingsModel {
        resizable: string
        vertical: string
        horizontal: string
    }
    class DragSettingsModel {
        draggable: string
    }
    class TransformSettingsModel {
        rotation: string
        translation: string
    }

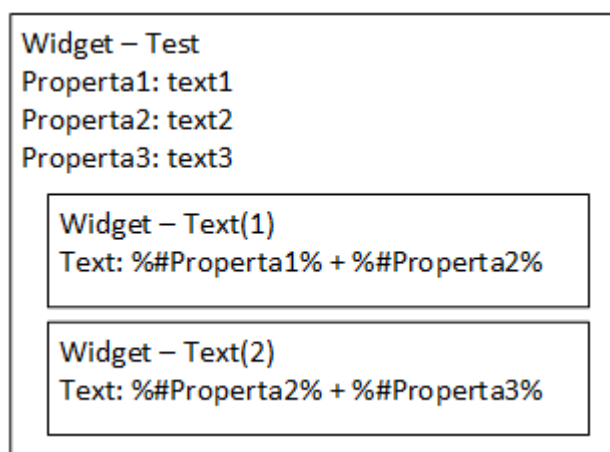
    ContentTemplateModel "1" -- "N" WidgetInstanceModel
    ContentTemplateModel "1" -- "N" TemplateGroupModel
    ContentTemplateModel "1" -- "1" ContentSettingsModel
    ContentTemplateModel "1" -- "N" PropertyModel
    ContentSettingsModel "1" -- "1" ToolboxSettingsModel
    ContentSettingsModel "1" -- "1" EditorSettingsModel
    EditorSettingsModel "1" -- "1" ResizeSettingsModel
    EditorSettingsModel "1" -- "1" DragSettingsModel
    EditorSettingsModel "1" -- "1" TransformSettingsModel
    AdditionalPropertyValueModel "1" -- "N" PropertyModel
  
```

### 5.3. Property

42

- Name – název property
- Value – hodnota zadaná do property, obsahuje prostý text, zástupné znaky, dotazy na server a podobně
- CalcValue – hodnota property s již vyhodnocenými daty
- DefaultValue – hodnota, která se zobrazuje, pokud property nedostane odpověď ze serveru
- Parents – list property. Obsahuje property, které byly vytvořeny na základě zástupných symbolů
- Childs – list property, z kterých byla vytvořena tato property

Kolekce Parents i childs vázou property k property ve vztahu M:N. Je to z toho důvodu, že nově vzniklá property s názvem jméno, může být vytvořena z dvou widgetů typu text. A každý z těchto widgetů typu text, respektive jejich property text může obsahovat i více zástupných symbolů než jenom zmiňované jméno. Tyto reference jsou potřeba pro propsání například dat ze serveru z parent property do child property. Na obrázku můžeme vidět dvě instance widgetu typu text, každá z těchto instancí má property text, která se odkazuje na dvě různé property nově tvořeného widgetu test.



Obrázek 37 - Ukázka M:N vazby

## 5.4. Vlastnosti

Každý z widgetů má své vlastnosti. V systému jsou definovány tři úrovně vlastností. V době vytvoření se widgetu přiřadí vlastnosti, které mají všechny widgety. Tyto vlastnosti jsou rozděleny do několika skupin a mají své výchozí hodnoty.

### Skupina General

- WidgetClass – CSS třída, která se aplikuje na widget.
- DisabledWidgetClass – CSS třída, která se aplikuje na widget, v případě, že je ve stavu disabled.
- Disabled – Pokud je widget v tomto stavu, negeneruje žádné události.
- Visible – Tato vlastnost určuje, jestli se widget zobrazí nebo ne.

### Skupina Location

- Top, Left, Width, Height – Základní vlastnosti určující pozici na stránce. Hodnoty lze zadávat v PX nebo %.
- Rotation, Translation – Vlastnosti widgetu umožňující jeho rotaci a transformaci podle různých os.

- Z-Index – Vlastnost určující pořadí renderování widgetů.

#### Skupina Tooltip

- Tooltip widget, Tooltip script, Track the mouse – skupina vlastností definující chování tooltipu.

#### Skupina MouseEvent

- Click, Over, Out – vlastnosti definující metody, které se budou volat v případě událostí

#### Skupina Security

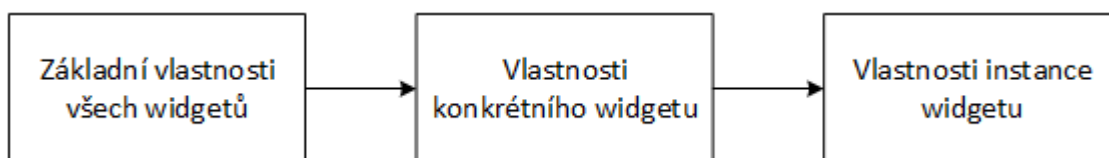
- SecurityVisibleFor, SecurityClickFor – vlastnosti, které upravují chování widgetu podle rolí aktuálně přihlášených operátorů

#### Skupina Load

- postLoad, preLoad – skripty, které se spustí před startem načítání všech widgetů a po jeho dokončení

V okamžiku, kdy je implementován nový widget, jsou vlastnosti doplněny o konkrétní vlastnosti nově vytvořeného widgetu. Nově vytvořený widget má taky možnost přepsat nějaké základní vlastnosti, pokud jim například chce definovat jinou výchozí hodnotu.

Při vložení nové instance widgetu se widgetu vytvoří všechny základní vlastnosti. Následně se vlastnostem přidají vlastnosti konkrétního widgetu, které byly zmíněny o odstavci výše. Všechny vlastnosti se pak dají následně přepsat v editoru.



Obrázek 38 - Vlastnosti widgetu

Nevýhodou tohoto řešení je, že když dojde k úpravě nějakého stávajícího widgetu, například odebráním nějaké jeho základní vlastnosti, tak widgety, které jsou již umístěny ve stránce tuto vlastnost pořád mají. Až nově vložené widgety se vkládají bez této již odstraněné vlastnosti.

## 5.5. Editace

Pro tvorbu a editaci stránek a widgetů je třeba editační režim. Pokud má uživatel dostatečné oprávnění, zobrazím mu v pravém horním rohu obrazovky editační tlačítko. Toto tlačítko odkazuje na globální proměnou a přepíná její hodnotu na true/false. Angular hlídá hodnotu této proměnné a reaguje na její změnu. Jelikož se aplikace používá v produkci, veškeré editační komponenty nejsou do první změny načtené, aby nezpomalovaly náběh aplikace a její samotný chod. Při prvním přepnutí do editačního režimu se tyto komponenty inicializují a zobrazí se editor.

Editor umožňuje vkládání nových widgetů z menu. Pro generaci menu využívám již hotovou kontrolku dxGrid [16]. Po vybrání položky z menu se vloží nová instance widgetu na výchozí souřadnice a nastaví se ji výchozí hodnoty.

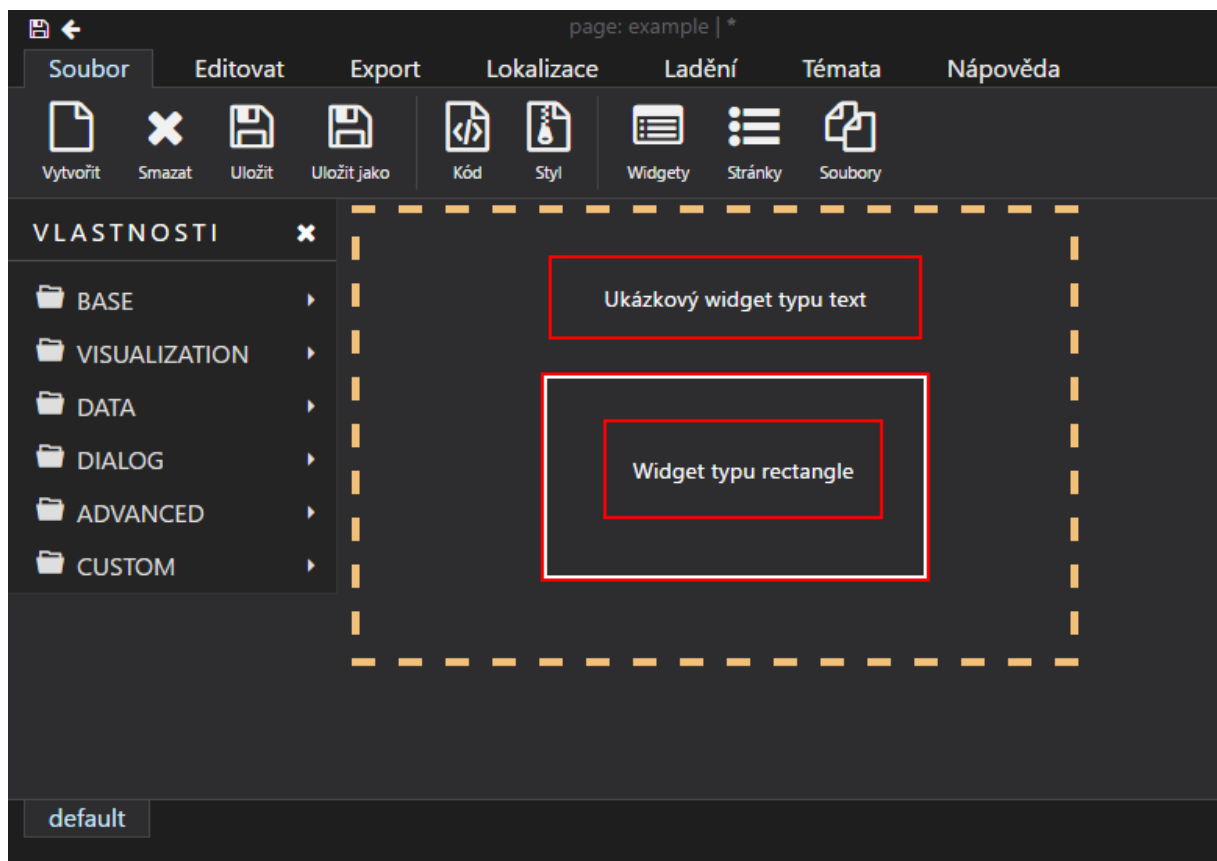
```

function setDraggable() {
    var dragSettings = widget.settings.editor.drag;
    if (dragSettings.draggable == "true") {
        element.draggable({
            cursor: 'move',
            snap: $rootScope.editor.borderSnapEnabled ? true : false,
            snapTolerance: 10,
            grid: $rootScope.editor.gridSnapEnabled ? [5, 5] : [1, 1],
        });
    }
}

```

*Zdrojový kód 8 - Ukázka metody setDraggable*

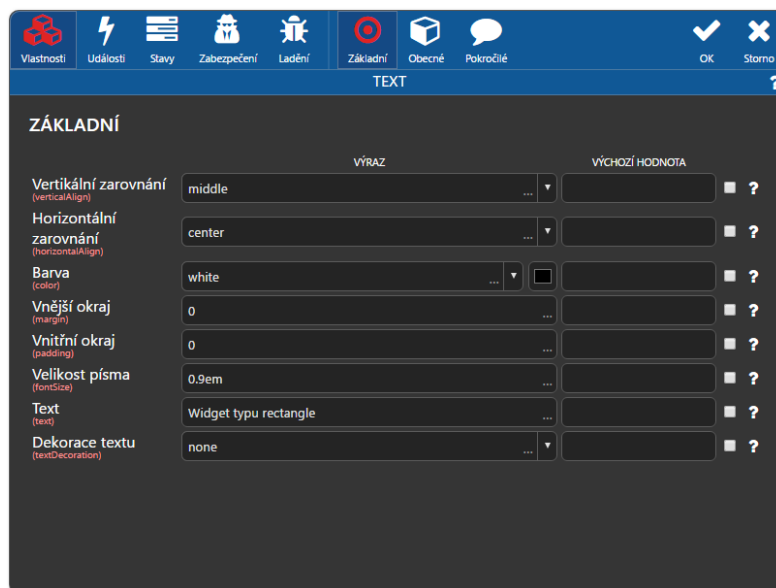
Každý z widgetů je obohacen o direktivu editable. Tato direktiva, stejně jako editor, hlídá globální hodnotu editace a po změně nastaví, nebo zruší svému widgetu určité vlastnosti. Pomocí knihovny jquery se widgetu podle jeho nastavení inicializuje vlastnost drag a resize. Při dvojkliku zobrazí editaci veškerých vlastností widgetu. Pro zobrazení dialogu se využívá služba dialog. Při kontextové nabídce zobrazí menu s možností editace, duplikace, kopírování do schránky, smazání widgetu a podobně. Při pohybu widgetu se potom automaticky aktualizují jeho vlastnosti top a left podle aktuálního umístění widgetu. Stejně tak i při změně velikostí dochází k aktualizaci vlastností.



*Obrázek 39 - Ukázka aplikace v editačním režimu*

Po dvojkliku, nebo výběru příslušné položky z kontextové nabídky se zobrazí editor vlastností vybraného widgetu. Pro zobrazení tohoto dialogu se opět využívá služba dialog. Každá z editovaných vlastností má v oblasti hodnoty vytvořený editor. Výchozí editor je text input a umožňuje zadat

jakoukoliv hodnotu. Editor ale podporuje více typů editorů. Každá z vlastností může mít přiřazený datasource. Na základě tohoto datasource objektu se poté vytvoří například listbox s hodnotami, které může vlastnost nabývat. Pro výběr barvy editor podporuje HTML5 input typu color. Tento input není bohužel podporovaný ve všech prohlížečích (IE). Uživatelé těchto prohlížečů jsou o tento editor ochuzeni. Pro složitější skripty editor podporuje zobrazit nad daným inputem pokročilejší editor s podporou syntaxe. Pro tyto účely jsem se rozhodl použít editor codemirror [17]. Tento editor nabízí spoustu funkcionalit a podporuje syntaxi všech potřebných jazyků (javascript, CSS, HTML). Do každé vlastnosti lze kromě statické hodnoty zadávat všechny typy již zmiňovaných výrazů reprezentujících lokální, globální a serverové proměnné.



Obrázek 40 - Ukázka editoru vlastností

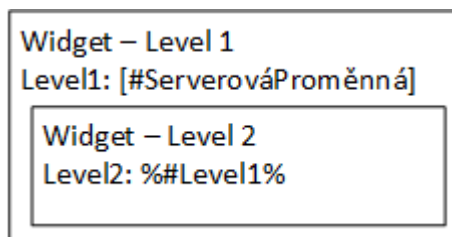
## 5.6. Jádru aplikace

Jako jádro bych označil vlastnost aplikace, která umožňuje nahradit všechny zástupné symboly, lokální, globální a serverové proměnné za skutečné hodnoty, případně nad nimi ještě zpracovat nějaký výraz.

### 5.6.1. Lokální proměnné

Lokální proměnné vytvářejí v aplikaci stromovou strukturu. Vždy, když u nějaké property dojde k aktualizaci hodnoty, je třeba provést i aktualizaci hodnot u property, které na tuto property odkazují.

Pokud u widgetu typu "Level 1" dojde ke změně hodnoty ze serveru, musí se tato hodnota propsat i do widgetu typu "Level 2". Tato aktualizace je zajištěná vazbou mezi property, které je zmíněna výše. Jakmile property provede aktualizaci svých hodnot, spustí rekurzivně aktualizaci všech svých potomků.

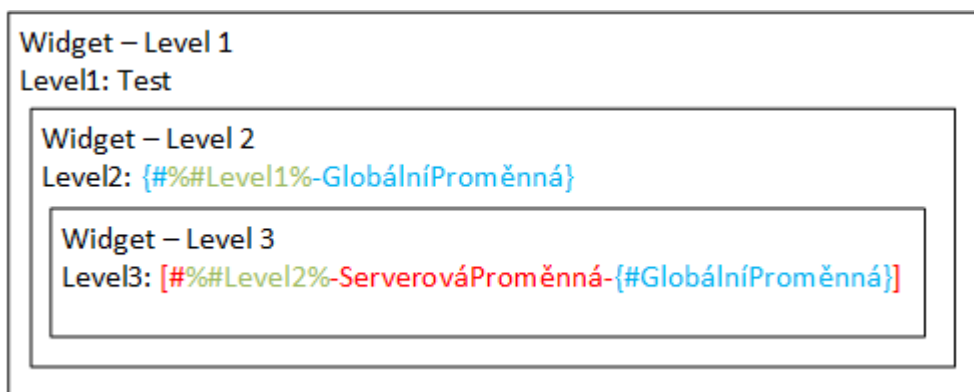


Obrázek 41 - Ukázka závislosti property

### 5.6.2. Globální a serverové proměnné

Globální a serverové proměnné fungují na podobném principu. Rozdíl je pouze v tom, že globální proměnné jsou řízeny uživatelskými skripty a serverové proměnné jsou řízeny nově přichozími daty ze serveru.

Všechny typy proměnných lze kombinovat do jedné syntaxe. Je ale třeba dodržet určitá pravidla. Lokální proměnné mohou obsahovat pouze text. Globální proměnné kromě textu mohou obsahovat také lokální proměnné. Serverové proměnné mohou obsahovat všechny typy proměnných. Na přiloženém obrázku můžete vidět, jak widget typu "Level 2" kombinuje lokální a globální proměnnou. Widget typu "Level 3" pak kombinuje všechny typy proměnných, a navíc využívá hodnotu z předchozí úrovně, která taky byla získána z globální proměnné.



Obrázek 42 - Kombinace proměnných

### 5.6.3. Aktualizace

Abych zajistil funkčnost popsanou v předchozím bodě, musí properta při každé aktualizaci znovu vše zkontrolovat, jestli nedošlo v jejich hodnotách ke změně.

Nejprve se musí najít lokální proměnné. Tyto proměnné lze jednoduše najít pomocí regex výrazu. Následně se musí všechny lokální proměnné projít, podle jejich názvu se najde adekvátní properta v "parent" widgetu a vezme se její hodnota. Tato hodnota se poznačí a dále se s ní pracuje.

---

```
var localVariables = this.value.toString().match(/%#(.*)%/g);  
var globalVariables = this.varExpression.toString().match(/#{(.*)}/g);  
var serverVariables = this.reqExpression.toString().match(/\[#(.*)\]/g);
```

---

Zdrojový kód 9 - Ukázka regex výrazů

Druhé v pořadí jsou globální proměnné. Tyto proměnné se hledají již nad upravenou hodnotou, které vznikla nahrazením z lokálních proměnných. Po nalezení globální proměnné, se celá properta uloží do služby variableLayer pod názvem nalezené proměnné. Jelikož se v průběhu aktualizace může stát, že se název proměnné změní, je třeba tento název porovnat s předchozím názvem a v případě změny je třeba tuto propertu ve variableLayeru přeregistrovat. Následně se z variableLayeru získá aktuální hodnota pro nalezenou proměnnou a hodnota se poznačí a bude se s ní dále pracovat.

Poslední v pořadí je serverová proměnná. Proces nahrazení je stejný jako v předchozím případě. Rozdíl je akorát v tom, že se odkazuje na dataLayer na místo variableLayeru. Registrace proměnné se potom odesílá na server jako požadavek na data.

Následně se zkontroluje, jestli nemá u property dojít k vyhodnocení skriptu. V takovém případě se pomocí služby evaluator hodnota vyhodnotí. Posledním krokem je spuštění aktualizace na všech "child" proprietách.

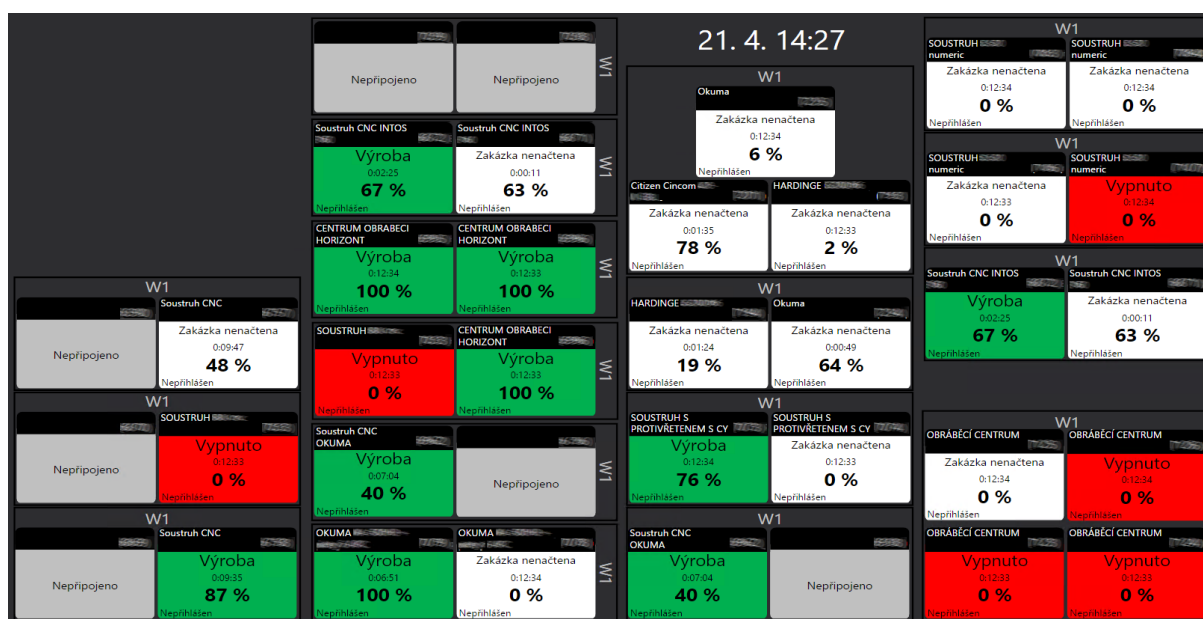


## 6. Nasazení aplikace

Aplikace je v současné době plně funkční a hojně se využívá u mnoha našich zákazníků. Možnosti využití aplikace jsou téměř neomezené. Webové prostředí a možnost psát vlastní skripty dělá z aplikace opravdu silný vývojový nástroj.

### 6.1. Přehledové obrazovky

Systém se dá také využít na přehledových obrazovkách pro zobrazení aktuálního stavu všech strojů v dílně. Je tak neocenitelným pomocníkem managementu, který tak okamžitě vidí, že některé ze strojů neběží na plný výkon a mohou tak okamžitě reagovat. V některých případech vytváříme i několik úrovní přehledových obrazovek, protože každá úroveň managementu potřebuje na své přehledové obrazovce vidět jinou úroveň detailů. Pro mistra v dílně jsou potřeba větší detaily než pro manažera v kanceláři.



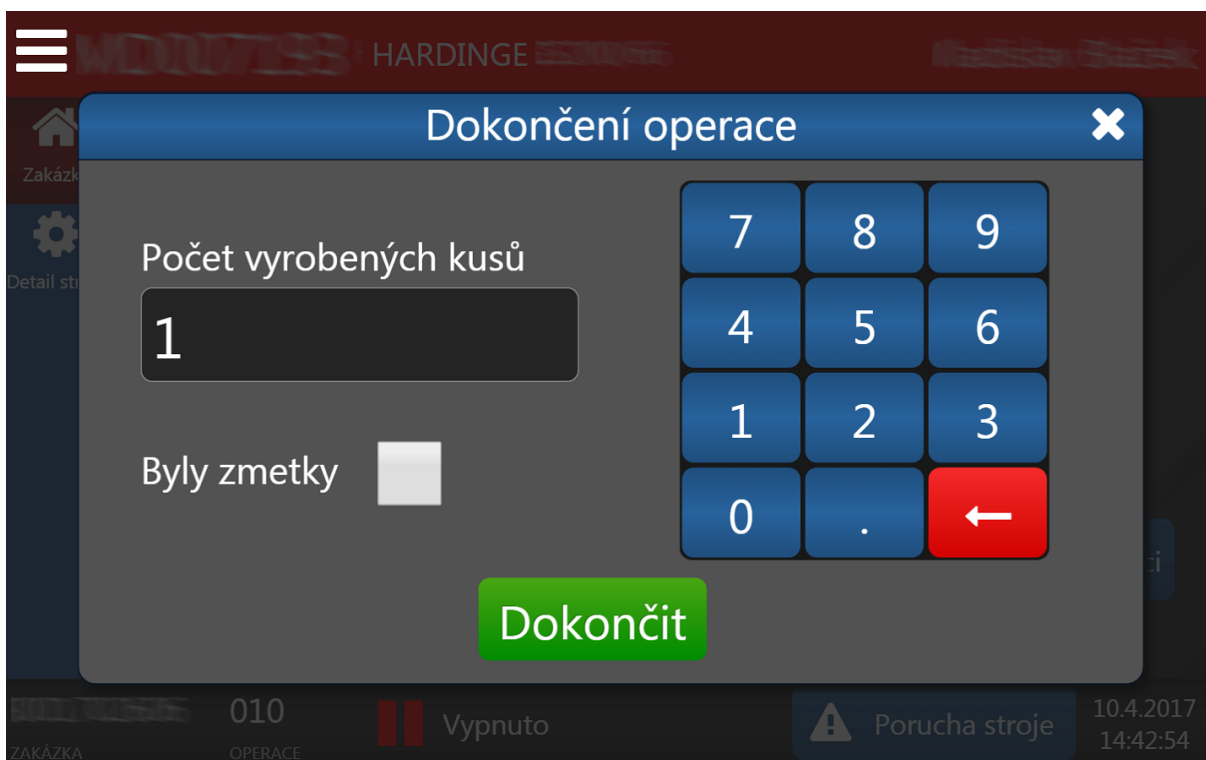
Obrázek 43 - Přehled dílny

### 6.2. Odvádění výroby

Hlavní využití má aplikace v samotné výrobě. Na dotykových terminálech slouží pro kontrolu a odvádění výroby na různých strojích. V průběhu zakázky kontrolujeme všechny vstupní a výstupní materiály pomocí čárových kódů a snažíme se tak eliminovat použití nesprávných komponent pro výrobu. Pokud operátor svou nepozorností použije špatnou vstupní komponentu, má to většinou za následek stáhnutí celé šarže výrobků. Taková chyba operátora může naše zákazníky stát nemalé peníze.

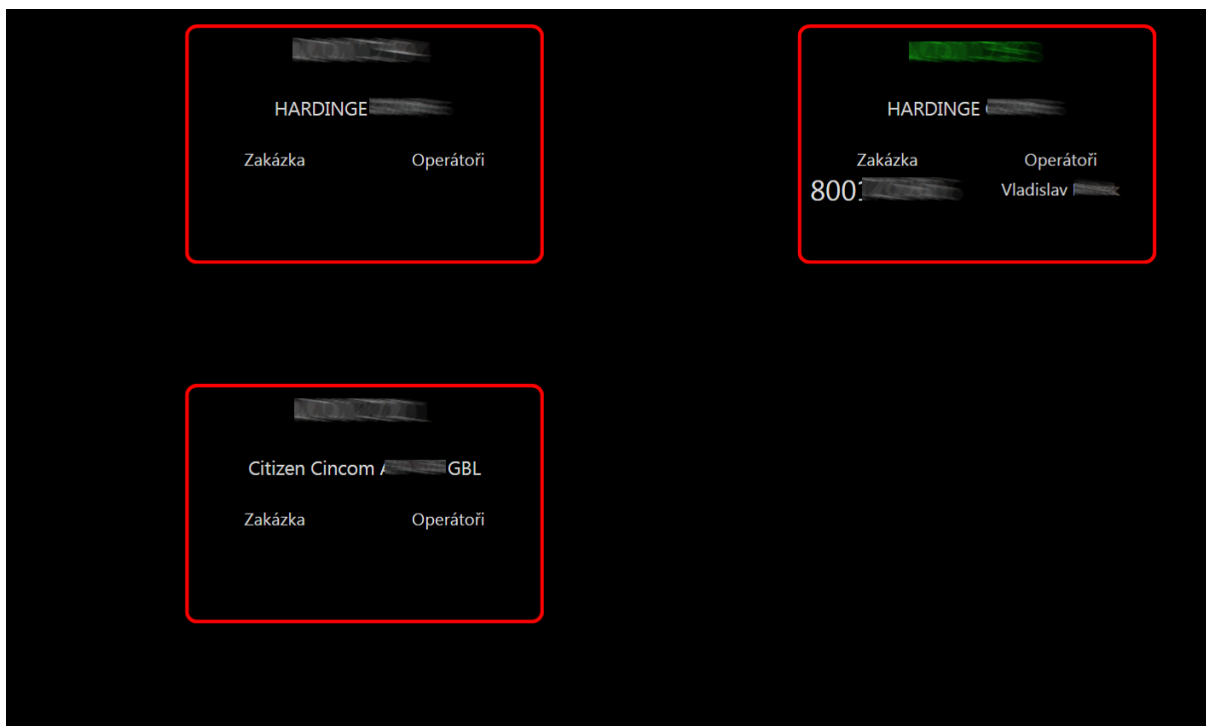
Pro sledování výroby potřebujeme mít nějaké základní informace o zakázce, od operátorů a také informace ze stroje. Proto, než operátor začne pracovat, vyžadujeme jeho přihlášení. Následně musí načíst zakázku, tím získáme potřebné časy a informace. Následně sledujeme využití stroje a žádáme operátora, aby nám řekl, proč zrovna stroj nejel. Na konci potom necháme operátora zadat informace o proběhlé operaci.

Důležité je si ale uvědomit, že náš systém samotnou výrobu nezefektivní. Pouze ukáže na rezervy v samotné výrobě nebo ve výrobním procesu. Odstranění těchto problémů pak přináší zlepšení výroby.



Obrázek 44 - Ukázka odvedení zakázky

Při instalaci systému do dílen je třeba zohlednit počet strojů. Umísťovat terminál vedle každého stroje by bylo finančně nákladné. Proto umožňujeme připojit k jednomu terminálu několik strojů.

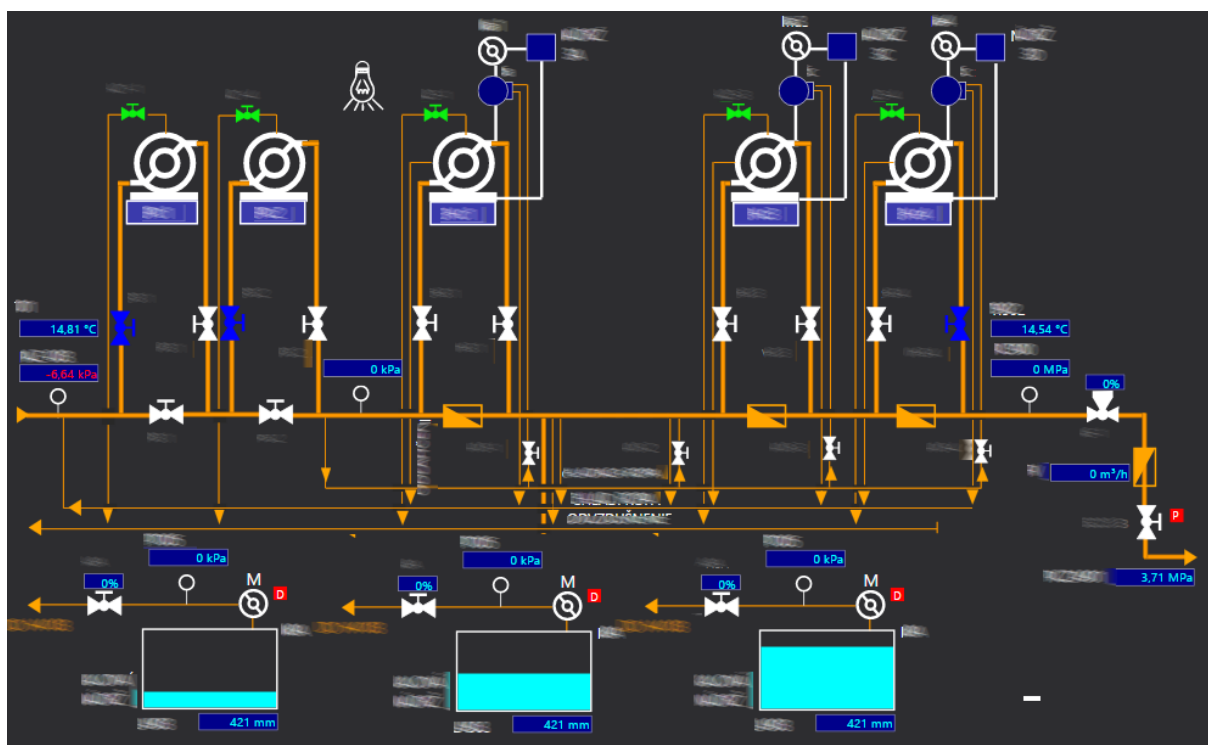


Obrázek 45 - Ukázka spořicí obrazovky

Operátor si pak v přehledové obrazovce vybere stroj, s kterým chce pracovat. Po nečinnosti se na terminálu opět zobrazí přehledová obrazovka, která slouží jako spořič, aby se zabránilo zbytečnému vypalování statického obrazu na LCD displejích.

### 6.3. Sledování technologie

V aplikaci je taky možné nakreslit celé technologické schéma různých rozvodů. V schématu lze potom sledovat různé informace. Můžeme zobrazit hladinu nádrže, stav ventilů, nebo třeba čerpadel. Také lze na základě výrazu určit, v kterém potrubí je nějaké médium a tuto informaci také na stránce zobrazit. Z čidel můžeme také zobrazovat rychlost průtoku, nebo teplotu. Teoreticky je možné technologie z takové přehledové obrazovky ovládat. Nicméně z důvodu bezpečnosti zatím tento scénář neumožňujeme.



Obrázek 46 - Ukázka potrubních rozvodů

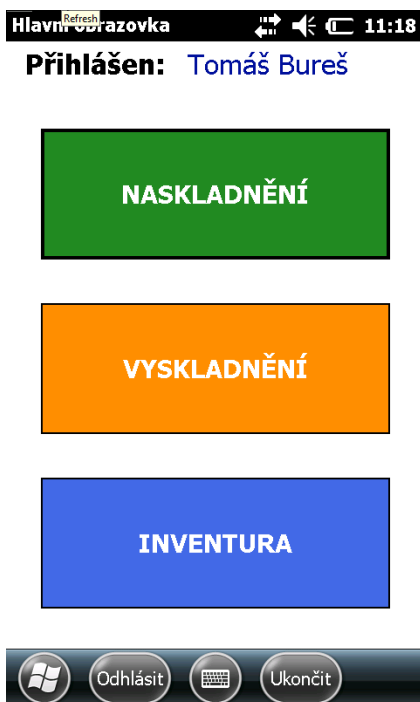
### 6.4. Mobilní terminály

Jelikož celá aplikace je postavena nad webovou technologií, může běžet na každém zařízení, které má k dispozici moderní webový browser. Díky tomu můžeme aplikaci využívat na moderních čtečkách značky zebra. Tyto čtečky běží na operačním systému android a mají tak plnou podporu webového browseru. Přes API této čtečky potom webový browser může komunikovat s periferiemi zařízení. Komunikace je obdobná jako komunikace přes nodejs, která je popsána výše.



Obrázek 47 - Moderní čtečka [18]

Díky těmto zařízením můžeme naši aplikaci využít pro implementaci skladového systému. Skladník za pomoci čtečky může zpracovávat příjemky, výdejky a dělat inventuru. Při překládání zboží potom může pomocí čárového kódu ověřit, jestli mu jde pod rukami správný kus. Při naskladnění skladníkovi pak přímo na displeji aplikace zobrazí číslo regálu.



Obrázek 48 - Ukázka aplikace na mobilním terminálu

## 7. Nedostatky, kam to posunout dál

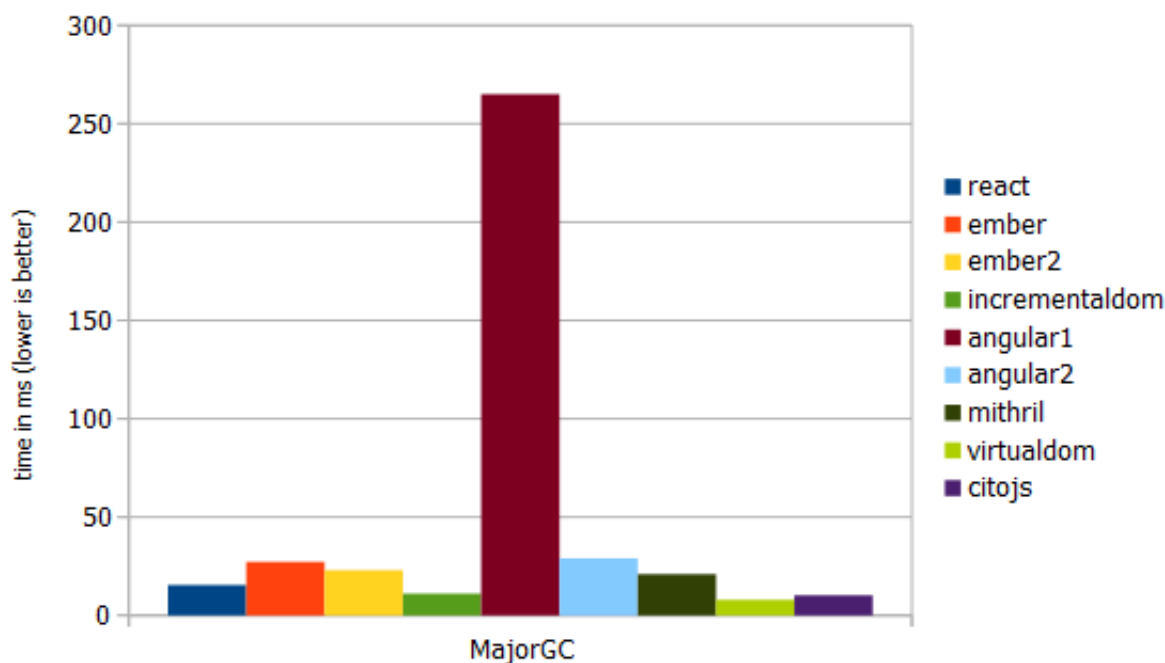
S fungováním aplikace jsme v celku spokojeni. Nicméně, určitě je pořád co zlepšovat, a i nějaké chyby se sem tam ještě objeví.

### 7.1. Problémy s výkonem

Aplikace se nenasazuje jenom na klasické PC, ale i na dotykové terminály, jejichž procesorový výkon a množství paměti je omezené. Proto jsem již několikrát musel řešit problémy s výkonem. Ve většině případech se mi podařilo najít nějaké chyby, které způsobovaly únik výkonu, ale i tak se už v některých rozsáhlejších aplikacích, se současnou implementací, dostávám k limitům hardwaru.

Řešením situace by mohlo být několik. Pro nové zákazníky již doporučuji nakupovat nové verze terminálu, které jsou výrazně rychlejší a mají větší množství paměti. Nicméně, toto řešení není vhodné pro stávající zákazníky, kteří mají skladem již desítky terminálů.

Další řešení, které zvažuji je přechod angularu na jeho novější verzi. Proto jsem začal experimentovat s novější verzí. Angular 2 [19] má oproti svému předchůdci v určitých scénářích razantní nárůst výkonu. Nicméně přechod na novou verzi by znamenal přepsání velkého množství řádků kódu, jelikož verze angularu nejsou zpětně kompatibilní. Od tohoto řešení mě ale odradilo oznámení další verze angularu, tentokrát s číslem 4 [20], a to pár měsíců po vydání verze 2. Pro ukázkou přikládám porovnání výkonnostních testů různých javascript frameworků.



Obrázek 49 - Porovnání výkonů js frameworků [21]

Třetí řešení problémů je částečné odstranění angularu z jádra aplikace. Angular by se dále využíval v celém editačním rozhraní aplikace. Toto řešení je dle mého názoru nejvhodnější. Nebylo by třeba přepisovat takové množství kódu, jako v předchozím případě. Implementace editační části by mohla být nedotčena a dále fungovat s použitím angularu. Důvod je ten, že na terminálech běží pouze jádro

aplikace a editace se provádí ve většině případů na našich strojích, na kterých problémy s výkonem nezaznamenáváme. Pro implementaci jádra bych použil pouze vlastní kód s využitím knihovny JQuery.

Existuje ještě další řešení, například využití HTML5 canvasu. Toto řešení je výhodné v tom, že je výkonné při použití téměř jakéhokoliv množství objektů, protože by se vykreslovaly pouze objekty, které jdou vidět. Rozdíl je v tom, že webový prohlížeč musí držet v paměti celý "Dokument object model". Toto řešení by ale obnášelo opuštění od HTML a webové aplikace. Nicméně, toto řešení by znamenalo úplné přepsání aplikace a začátek od nuly, takže jsem ho chtěl zmínit, jako alternativní postup. Kdybych začínal znovu, určitě bych ho zvažoval.

## 7.2. Nápady na vylepšení

Aplikace nyní obsahuje tři entity. První jsou stránky a widgety, druhá jsou skripty a styly a třetí jsou všechny ostatní soubory. Každá z těchto entit se ukládá do samostatných tabulek v různých formátech. Widgety, stránky, skripty a styly podporují pouze plochou strukturu. Jinými slovy, nejdou třídit po složkách ve stromovém systému, tak jako u souborů. Do budoucna bych chtěl všechny tyto entity ukládat v jedné stromové struktuře a na základě přípony obsah souboru zobrazit v příslušném editoru. Jelikož se aplikace využívá jako vývojové prostředí, umožňovalo by to lepší a přehlednější třídění souborů.

Jak jsem již zmínil, na dotykových terminálech se potýkáme s kolísáním výkonu. Ne vždy je tento problém na naší straně. Zjistili jsme, že v některých případech je chyba v samotném HW a z nějakého důvodu je terminál podtaktovaný a má nižší výkon než ostatní kusy. Pro detekci takových terminálů využíváme "Fishbowl" test. Tento test spočívá v tom, že přidává objekty do webového prostředí a testuje, kolik objektů zvládne bez poklesu FPS. Název je odvozen od toho, že objekty jsou rybičky, které plavou v akváriu. Návrhem pro zlepšení by bylo integrovat tento test přímo do aplikace.



Obrázek 50 - Ukázka FishBowl testu [22]

## 8. Závěr

Cílem práce bylo vytvořit webové rozhraní pro zobrazení dat. Projekt jsem pojal jako takzvanou "single page" aplikaci. Aplikace pomocí jednoduchého editoru umožňuje efektivně tvořit a editovat různé přehledové obrazovky, nebo aplikace na terminály. Projekt je přínosem jak pro naši firmu, protože šetří čas vývojářů při tvoření aplikací pro zákazníky, taktéž je přínosem pro zmíněné zákazníky. Sledování výroby a odvádění zakázek přináší našim zákazníkům důležité informace. Na základě těchto informací pak mohou naši zákazníci optimalizovat svůj výrobní proces.

Na aplikaci pracuji již druhým rokem. V začátcích se v průběhu nasazování objevovaly chyby v jádru aplikace a musely být rychle opraveny. Při nasazení aplikace jsem měl obavy ze stability systému. Sice se v průběhu objevily nějaké problémy s výkonem, které jsme prozatím odstranili, ale kromě zmíněných problémů s výkonem je aplikace spolehlivá. V současné době je již aplikace stabilní a chyby se vyskytují jen velmi zřídka. Na vývoji aplikace se pořád pracuje a na základě čím dál větších zkušeností z nasazení a integrace u zákazníků se rozšiřují základní funkce aplikace.

Při práci na tomto projektu jsem pracoval s velkým množstvím technologií a získal nemalé množství zkušeností, ať už to byla práce s webovým serverem, nebo implementace samotné aplikace za pomoci různých frameworků. Také jsem si vyzkoušel práci s nodejs serverem a několika jeho moduly, jako například socket.io, který umožňuje komunikaci přes websockety. Určitě musím taky zmínit SignalR technologii, kterou využíváme pro komunikaci s aplikačním serverem.

## 9. Reference

1. MacDonald, Matthew. *ASP.NET 4 a C# 2010*. místo neznámé : Zoner Press, 2011. 9788074131318.
2. Alexis Goldstein, Estelle Weyl. *HTML5 a CSS3 pro webové designéry*. místo neznámé : Zoner Press, 2011. 9788074131660.
3. Refsnes Data. CSS Tutorial. *W3Schools*. [Online] [Citace: 24. 4 2017.] <https://www.w3schools.com/css/>.
4. Nguyen, Don. *Node.js Okamžitě*. místo neznámé : COMPUTER PRESS, 2016. 9788025148204.
5. *Socket.io*. [Online] [Citace: 24. 4 2017.] <https://socket.io/>.
6. Kaazing ©. About HTML5 WebSocket. *Websocket*. [Online] [Citace: 24. 4 2017.] <https://www.websocket.org/aboutwebsocket.html>.
7. © 1998-2017 Developer Express Inc. *DevExpress*. [Online] [Citace: 24. 4 2017.] <https://www.devexpress.com/>.
8. Aguilar, Jose M. *SignalR Programming in Microsoft ASP.NET*. místo neznámé : Microsoft Press Corp., 2014. 9780735683884.
9. *WireShark*. [Online] [Citace: 24. 4 2017.] <https://www.wireshark.org/>.
10. Google ©2010-2017. Superheroic JavaScript MVW Framework. *AngularJS*. [Online] [Citace: 24. 4 2017.] <https://angularjs.org/>.
11. —. Dependency Injection. *AngularJS*. [Online] [Citace: 24. 4 2017.] <https://docs.angularjs.org/guide/di>.
12. Css Bootstrap. *Bootstrap*. [Online] [Citace: 24. 4 2017.] <http://getbootstrap.com/css/>.
13. Jonathan Chaffer, Karl Swedberg. *Mistrovství v jQuery*. místo neznámé : Computer Press, 2013. 9788025141038.
14. Pocklington, Robert. jQuery.Hotkeys. *Github*. [Online] [Citace: 24. 4 2017.] <https://github.com/jeresig/jquery.hotkeys>.
15. Philipp, Jan. Angular-Translate. *Github*. [Online] [Citace: 24. 4 2017.] <https://github.com/angular-translate/angular-translate>.
16. © 2011-2017 Developer Express Inc. *DevExtreme*. [Online] [Citace: 24. 4 2017.] <https://js.devexpress.com/>.
17. *CodeMirror*. [Online] [Citace: 24. 4 2017.] <https://codemirror.net/>.
18. ©2017 ZIH Corp and/or its affiliates. TC8000 TOUCH MOBILE COMPUTER. *Zebra*. [Online] [Citace: 24. 4 2017.] <https://www.zebra.com/us/en/products/mobile-computers/handheld/tc8000.html>.
19. Google ©2010-2017. One framework. *Angular 2*. [Online] [Citace: 24. 4 2017.] <https://angular.io/>.



20. Fluin, Stephen. *Angular 4*. [Online] [Citace: 24. 4 2017.] <http://angularjs.blogspot.cz/2017/03/angular-400-now-available.html>.
21. Peyrott, Sebastián. Virtual DOM vs Angular 1 & 2 vs Others. *Auth0*. [Online] 07. 1 2016. [Citace: 24. 4 2017.] <https://auth0.com/blog/more-benchmarks-virtual-dom-vs-angular-12-vs-mithril-js-vs-the-rest/>.
22. *FishBowl*. [Online] [Citace: 24. 4 2017.] <https://testdrive-archive.azurewebsites.net/performance/fishbowl/>.

## Příloha A. Obsah přiloženého DVD

- Text diplomové práce
- Zdrojové kódy aplikace